Chapter 1

Graphs

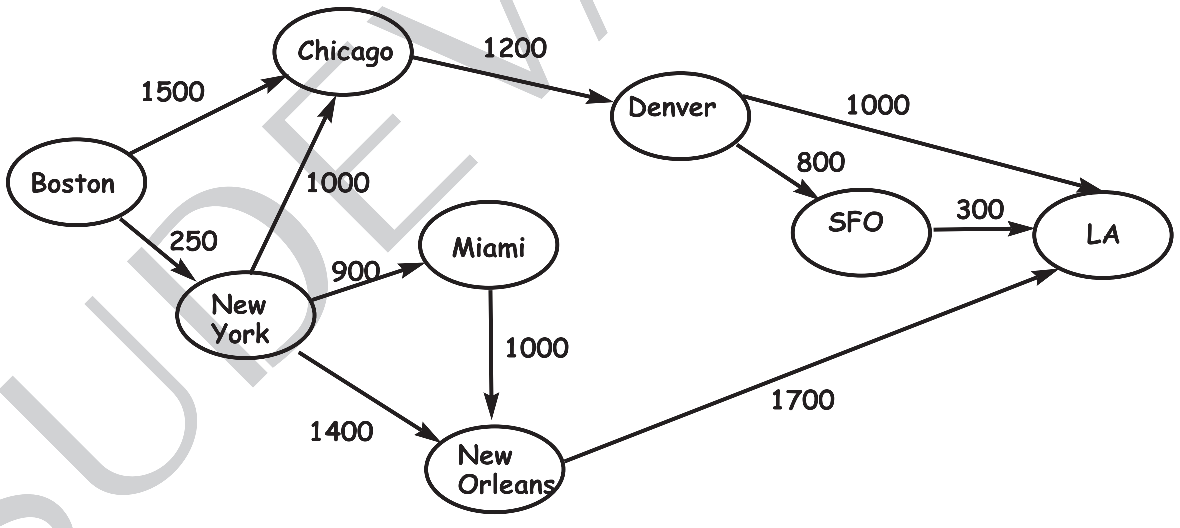
1.1 Introduction

1.2 Graph applications

1.2.1 Transportation problem

From	To	Miles
SFO	LA	300
Miami	New Orleans	1000
Boston	Chicago	1500
Boston	New York	250
Denver	SFO	800
Chicago	Denver	1200
New York	Miami	900
New York	Chicago	1000
New York	New Orleans	1400
Denver	LA	1000
New Orleans	LA	1700

What is the best way to fly from Boston to LA ?



Boston -> New York --> New Oreleans --> LA = 3350

Figure 1.1: What is the best way to fly from Boston to LA?

1.2.2 Minimum connector problem

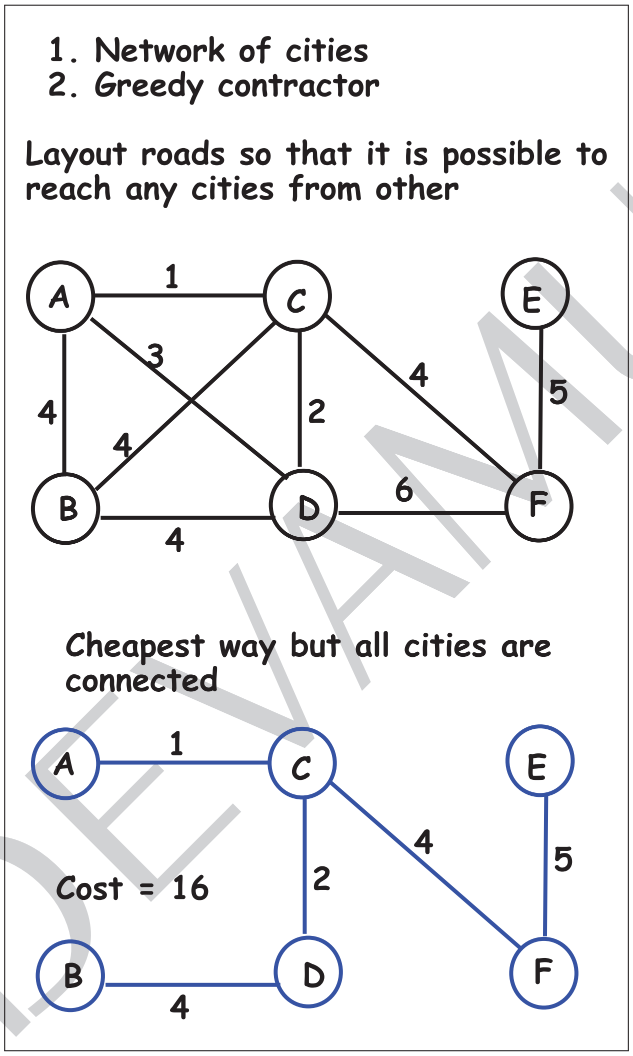

1. Network of cities
2. Greedy contractor
Layout roads so that it is possible to reach any cities from other
A
B
C
D
E
F
1
3
4
4
2
4
5
4
6
Cheapest way but all cities are connected
Cost = 16
A
B
C
D
E
F
1
2
4
5
4

Figure 1.2: Laying cheapest road

1.2.3 Scheduling problem

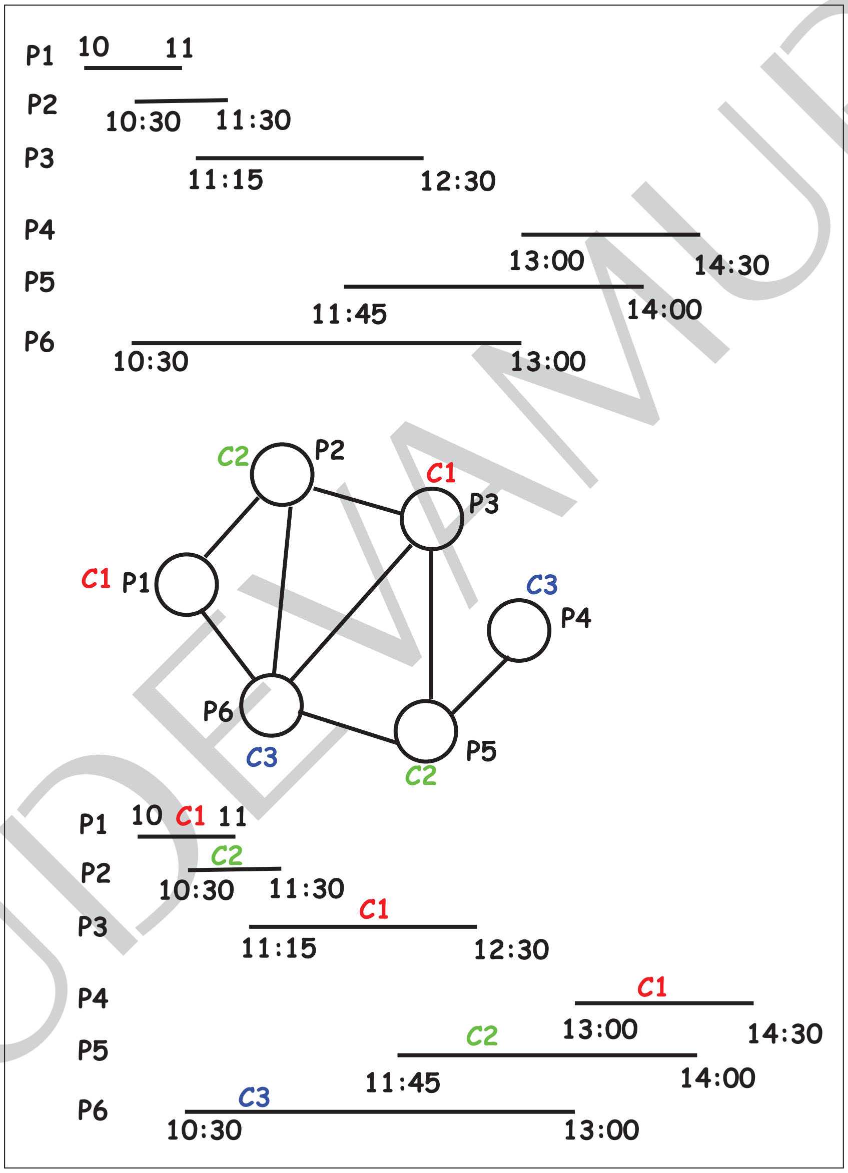


Figure 1.3: Minimum channels required to broadcast seven programs

1.2.4 Activity network or Topological sorting problem

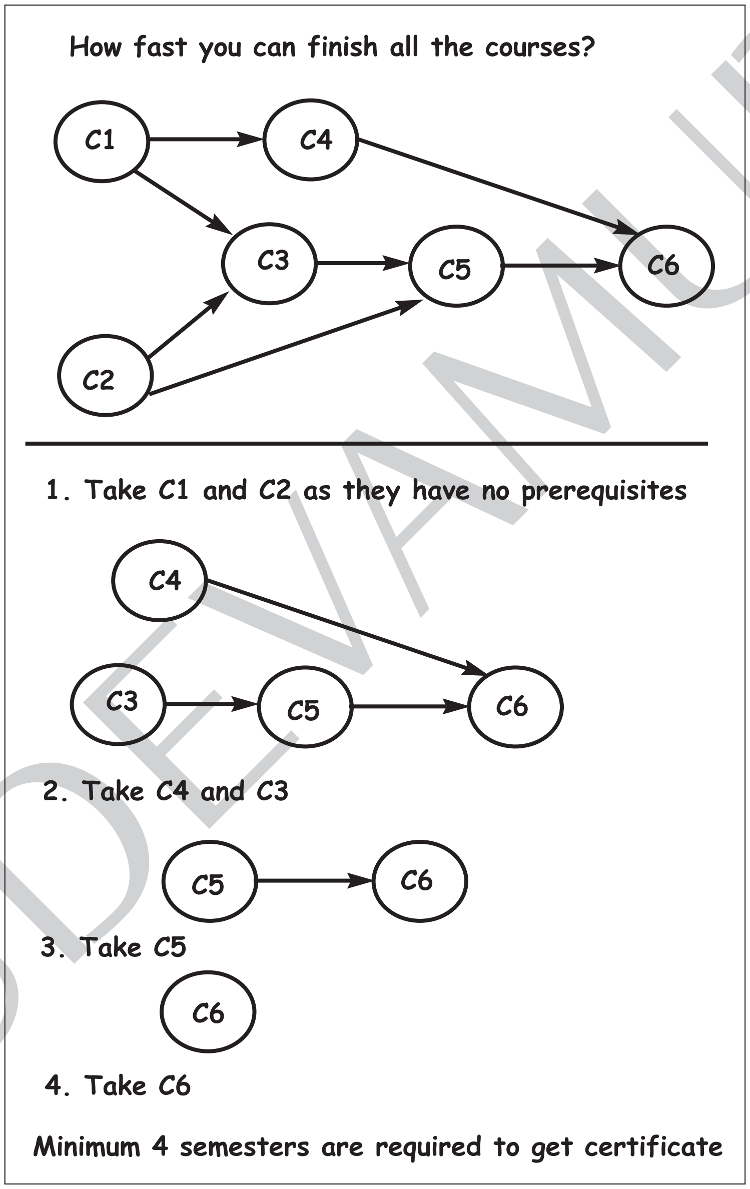

How fast you can finish all the courses?
C1
C4
C3
C5
C6
C2
1. Take C1 and C2 as they have no prerequisites
C4
C3
C5
C6
2. Take C4 and C3
C5
C6
3. Take C5
C6
4. Take C6
Minimum 4 semesters are required to get certificate

Figure 1.4: Completing courses in an university

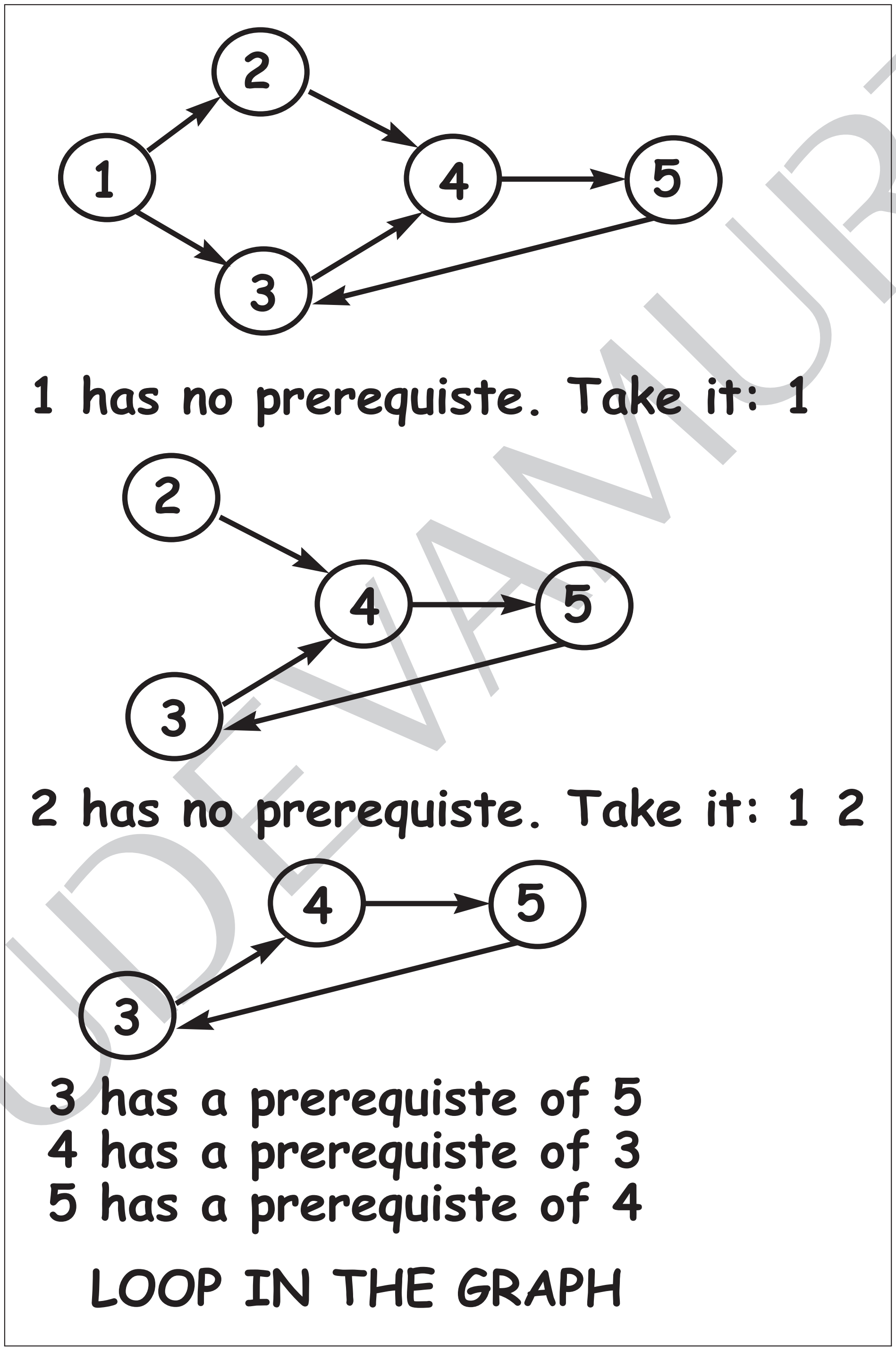


Figure 1.5: Impossible to complete courses

1.2.5 Critical path analysis

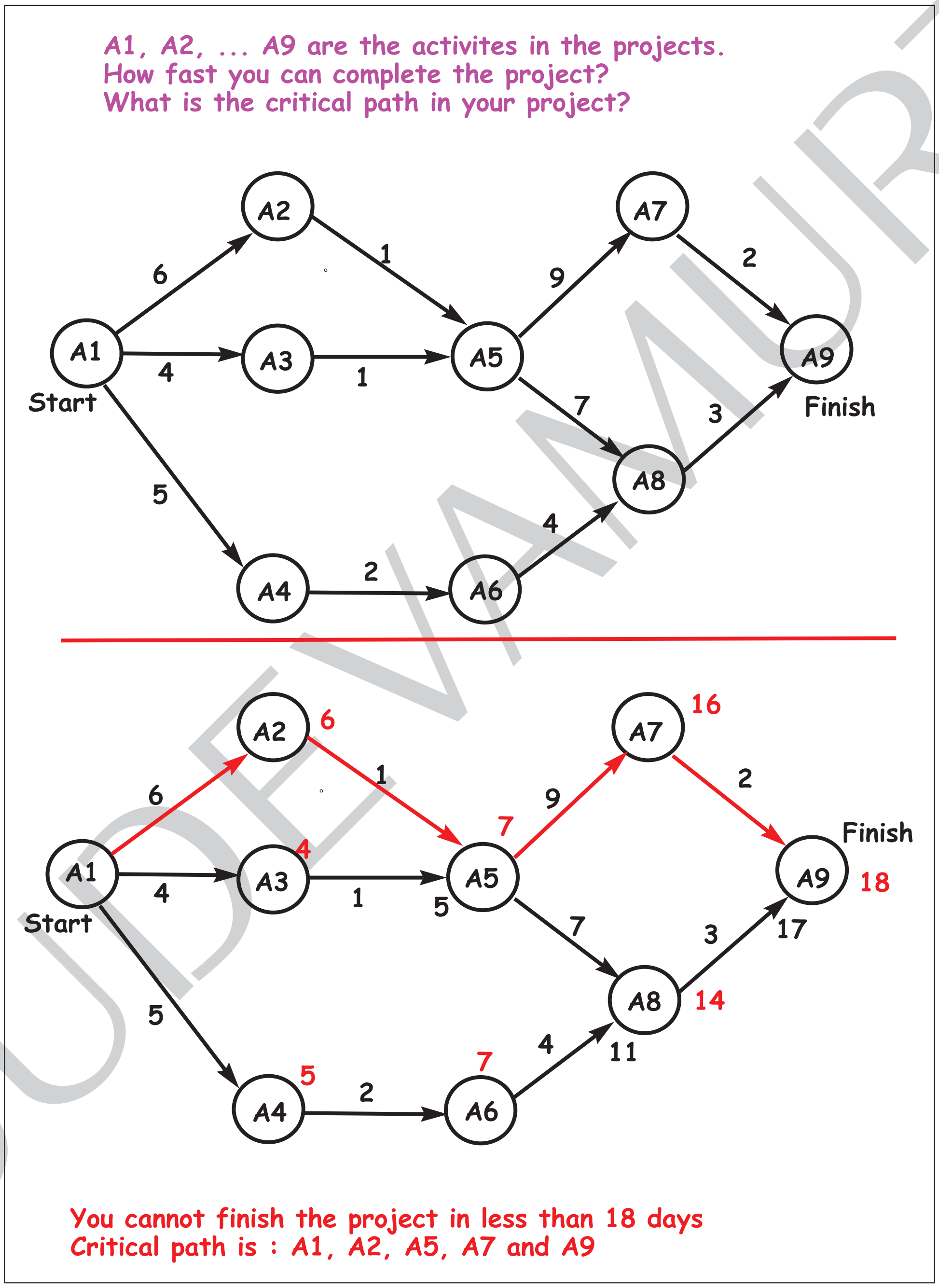
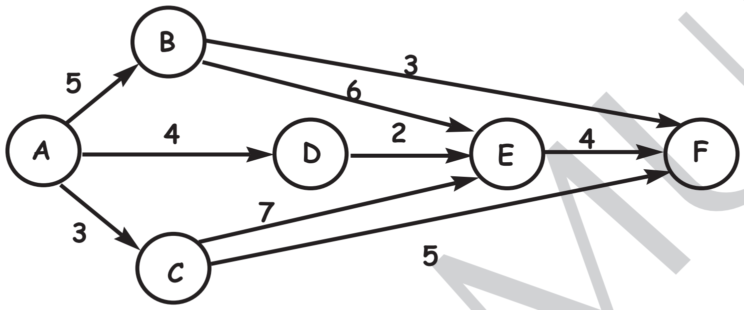

A1, A2, ... A9 are the activites in the projects.
How fast you can complete the project?
What is the critical path in your project?
A1
A2
A3
A4
A5
A6
A7
A8
A9
Start
Finish
6
4
5
1
1
2
9
7
2
3
4
16
7
5
18
17
14
11
You cannot finish the project in less than 18 days
Critical path is : A1, A2, A5, A7 and A9

Figure 1.6: Critical path of a project

1.2.6 Flow problem

Cities that are connected by pipelines
Number on edges represents maximum oil that can be sent
Goal to ship maximum oil as possible from source to destination
It is not possible to store oil en route.



A possible flow from A to F which ships 8 units in all.
Is this the best that can be done?

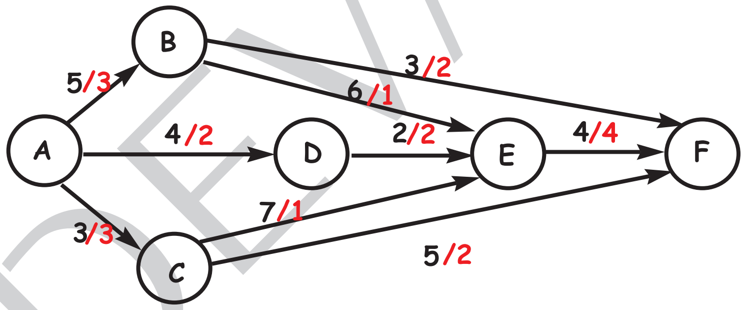
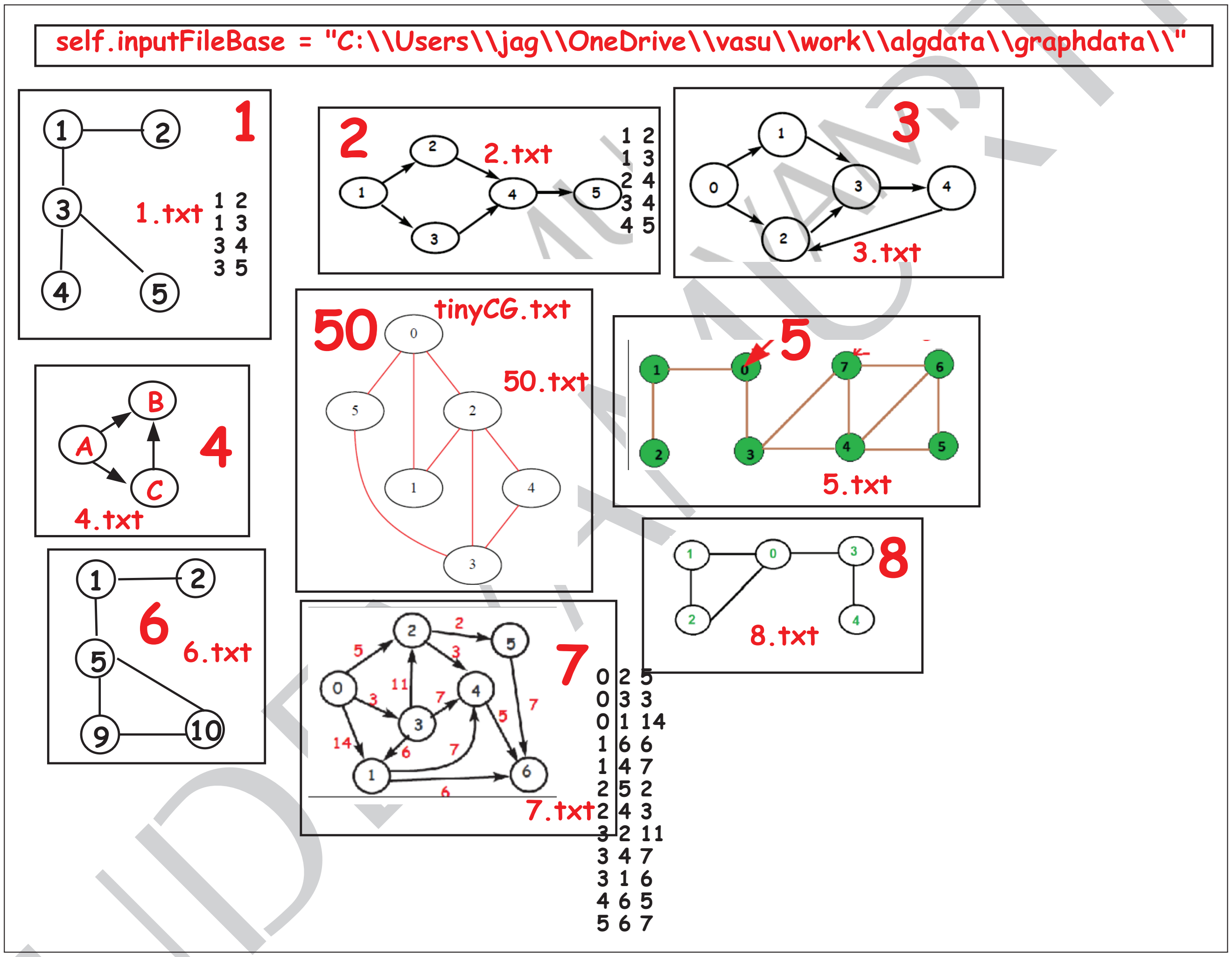


Figure 1.7: Maximum flow possible

1.3 Graph examples



```
self.inputFileBase = "C:\\Users\\jag\\OneDrive\\vasu\\work\\algdata\\graphdata\\"
```

```
1.txt
1 2
1 3
3 4
3 5
```

```
2.txt
1 2
1 3
2 4
3 4
4 5
```

```
7.txt
0 2 5
0 3 3
0 1 14
1 6 6
1 4 7
2 5 2
2 4 3
3 2 11
3 4 7
3 1 6
4 6 5
5 6 7
```

Figure 1.8: Various graphs

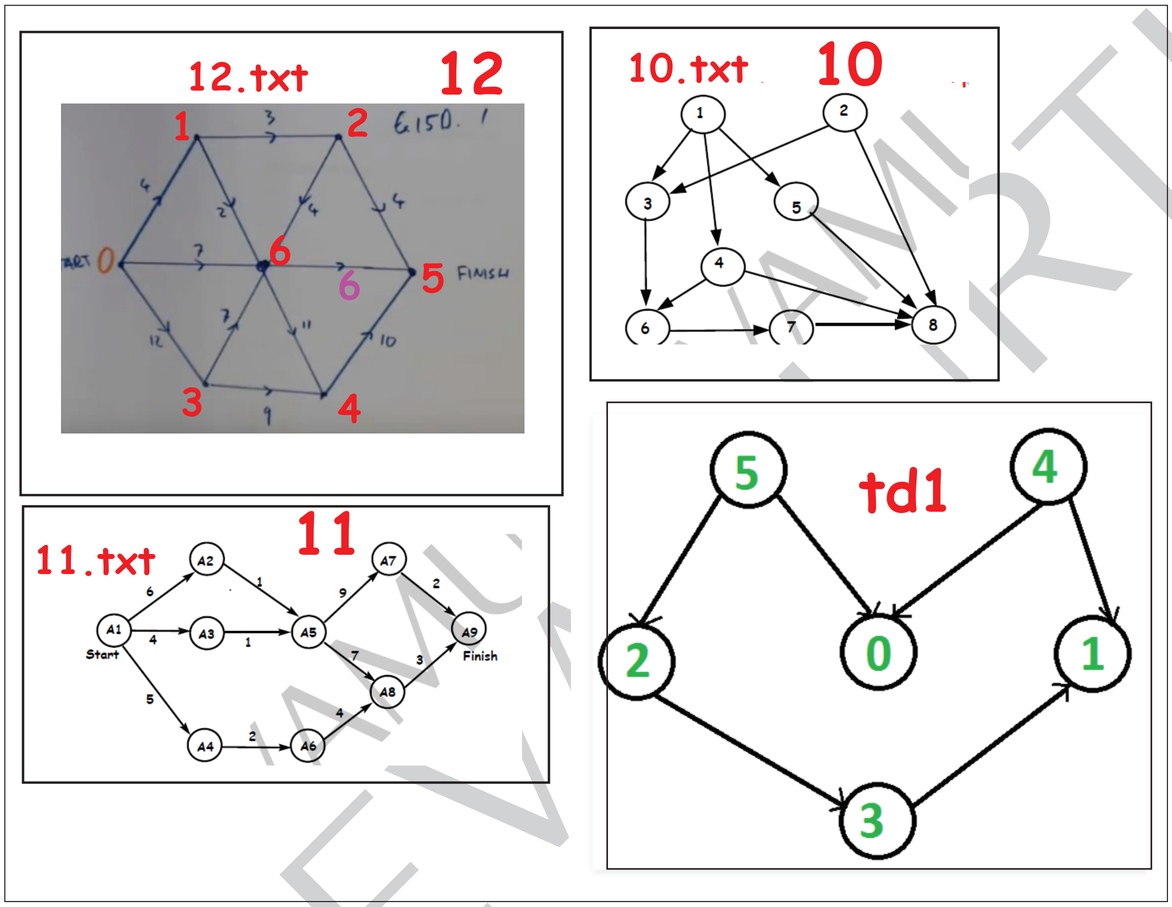


Figure 1.9: Various graphs

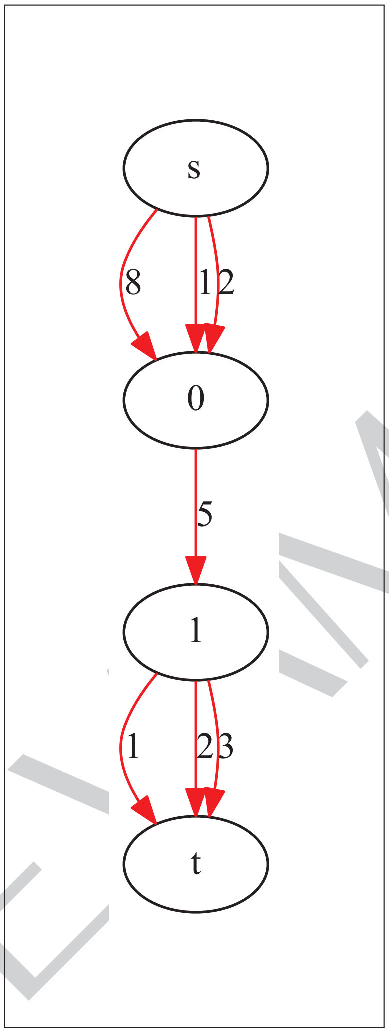
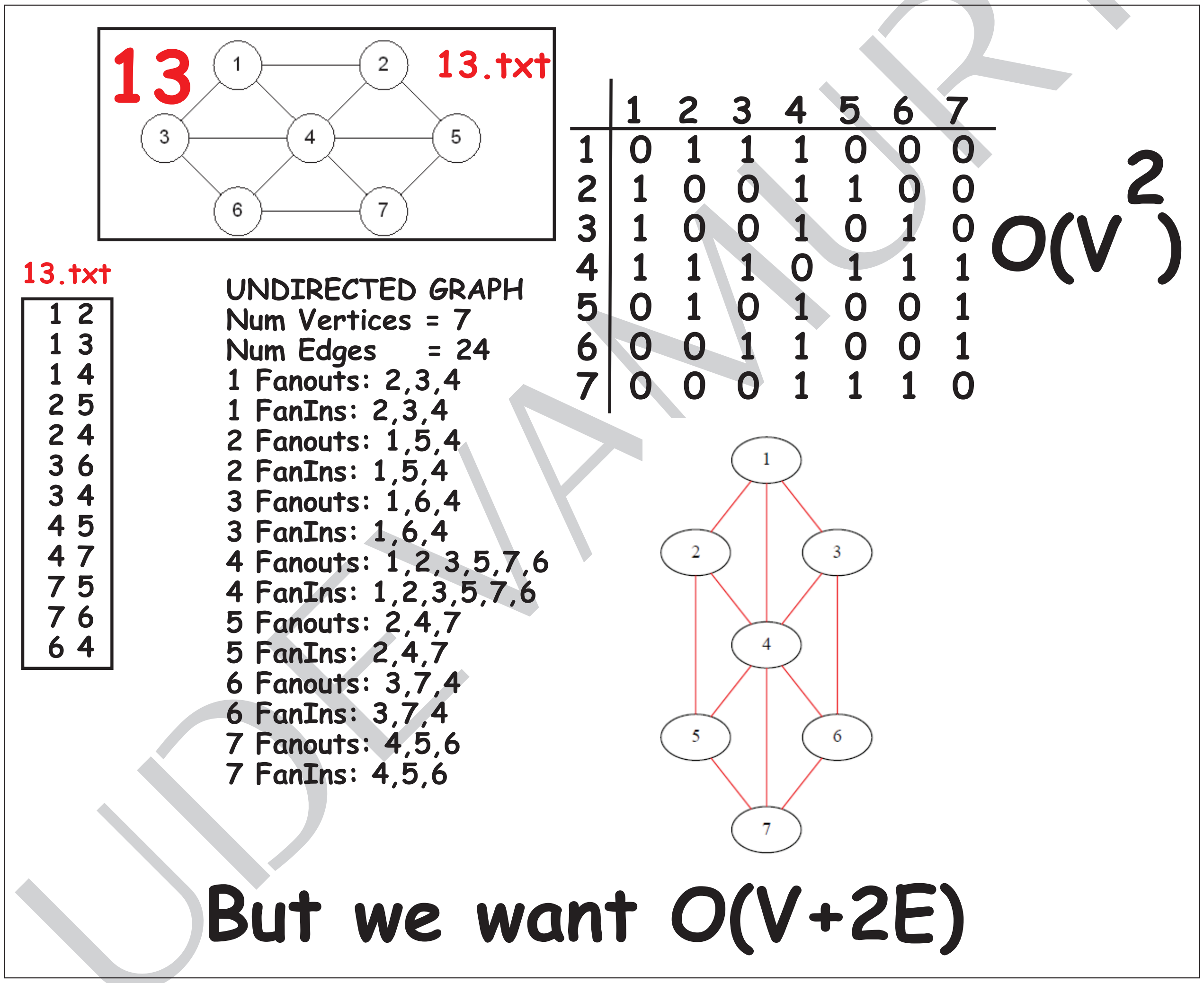


Figure 1.10: Parallel edges

1.4 Graph representation using matrices

1.4.1 Undirected graph representation



```
1 2
1 3
1 4
2 5
2 4
3 6
3 4
4 5
4 7
7 5
7 6
6 4
```

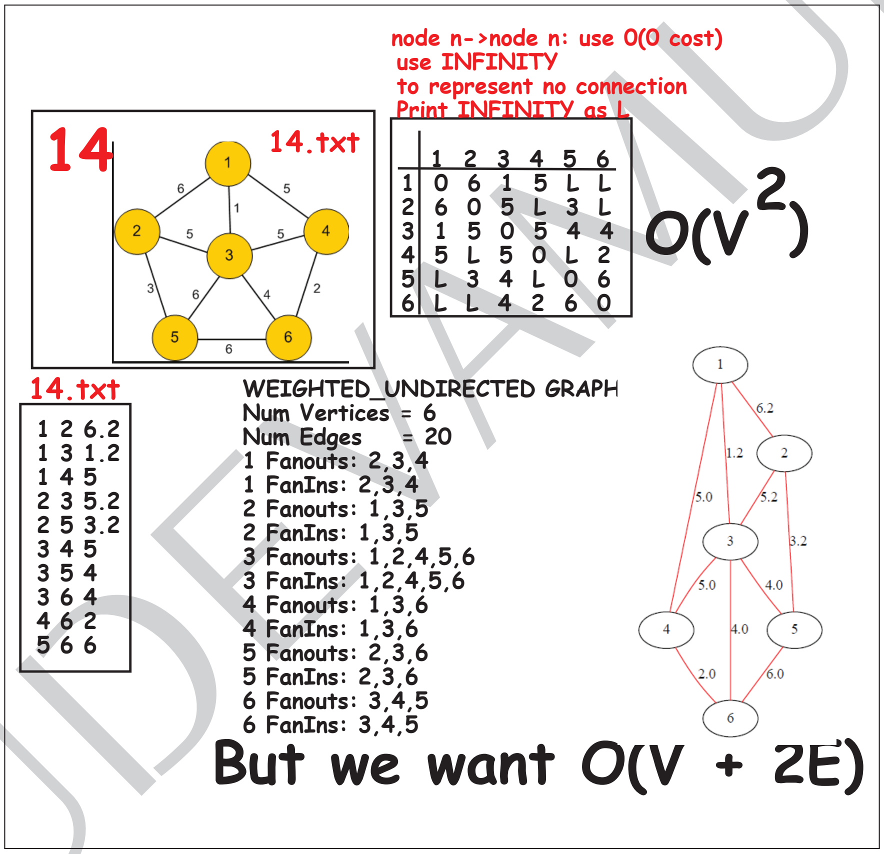
```
UNDIRECTED GRAPH
Num Vertices = 7
Num Edges    = 24
1 Fanouts: 2,3,4
1 FanIns: 2,3,4
2 Fanouts: 1,5,4
2 FanIns: 1,5,4
3 Fanouts: 1,6,4
3 FanIns: 1,6,4
4 Fanouts: 1,2,3,5,7,6
4 FanIns: 1,2,3,5,7,6
5 Fanouts: 2,4,7
5 FanIns: 2,4,7
6 Fanouts: 3,7,4
6 FanIns: 3,7,4
7 Fanouts: 4,5,6
7 FanIns: 4,5,6
```

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
3 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
4 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
5 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
6 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
7 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{array}
\quad O(V^2)$$

But we want O(V+2E)

Figure 1.11: Representation of an undirected graph

1.4.2 Undirected weighted graph representation



node n->node n: use 0(0 cost)
use INFINITY
to represent no connection
Print INFINITY as L

14

14.txt

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 6 & 1 & 5 & L & L \\ 2 & 6 & 0 & 5 & L & 3 & L \\ 3 & 1 & 5 & 0 & 5 & 4 & 4 \\ 4 & 5 & L & 5 & 0 & L & 2 \\ 5 & L & 3 & 4 & L & 0 & 6 \\ 6 & L & L & 4 & 2 & 6 & 0 \end{array}$$

$O(V^2)$

14.txt

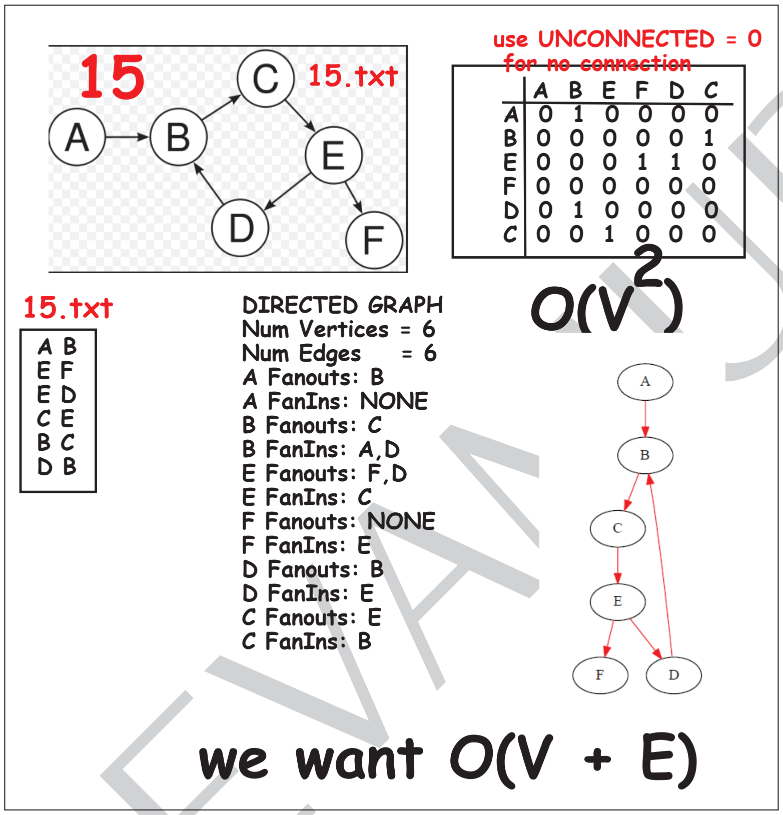
```
1 2 6.2
1 3 1.2
1 4 5
2 3 5.2
2 5 3.2
3 4 5
3 5 4
3 6 4
4 6 2
5 6 6
```

```
WEIGHTED_UNDIRECTED GRAPH
Num Vertices  = 6
Num Edges     = 20
1 Fanouts: 2,3,4
1 FanIns: 2,3,4
2 Fanouts: 1,3,5
2 FanIns: 1,3,5
3 Fanouts: 1,2,4,5,6
3 FanIns: 1,2,4,5,6
4 Fanouts: 1,3,6
4 FanIns: 1,3,6
5 Fanouts: 2,3,6
5 FanIns: 2,3,6
6 Fanouts: 3,4,5
6 FanIns: 3,4,5
```

But we want $O(V + 2E)$

Figure 1.12: Representation of an undirected weighted graph

1.4.3 Directed graph representation



	A	B	E	F	D	C
A	0	1	0	0	0	0
B	0	0	0	0	0	1
E	0	0	0	1	1	0
F	0	0	0	0	0	0
D	0	1	0	0	0	0
C	0	0	1	0	0	0

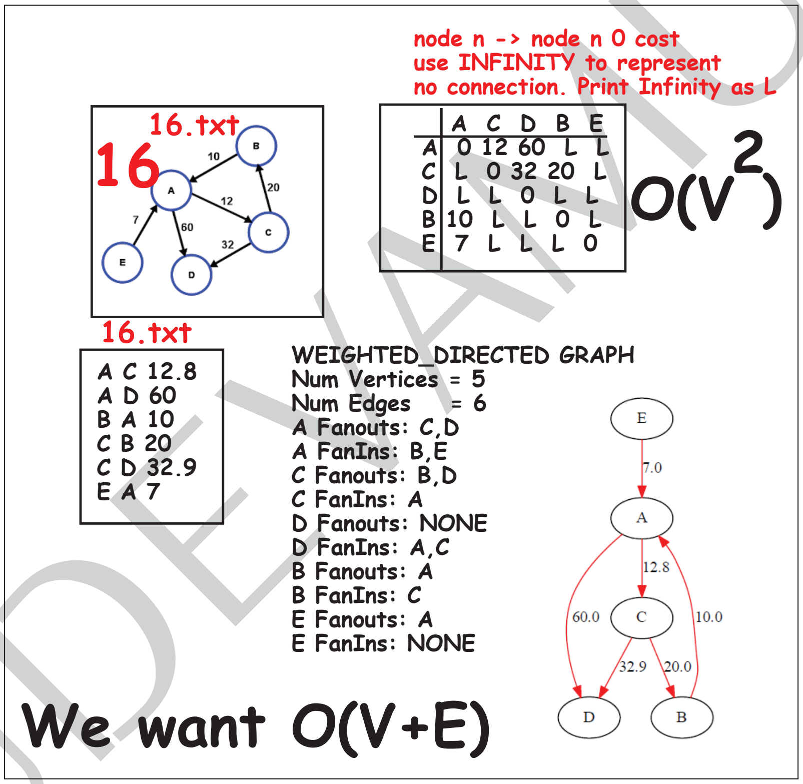
15.txt

```
A B
E F
E D
C E
B C
D B
```

```
DIRECTED GRAPH
Num Vertices = 6
Num Edges    = 6
A Fanouts: B
A FanIns: NONE
B Fanouts: C
B FanIns: A,D
E Fanouts: F,D
E FanIns: C
F Fanouts: NONE
F FanIns: E
D Fanouts: B
D FanIns: E
C Fanouts: E
C FanIns: B
```

Figure 1.13: Representation of a directed graph

1.4.4 Directed weighted graph representation



	A	C	D	B	E
A	0	12	60	L	L
C	L	0	32	20	L
D	L	L	0	L	L
B	10	L	L	0	L
E	7	L	L	L	0

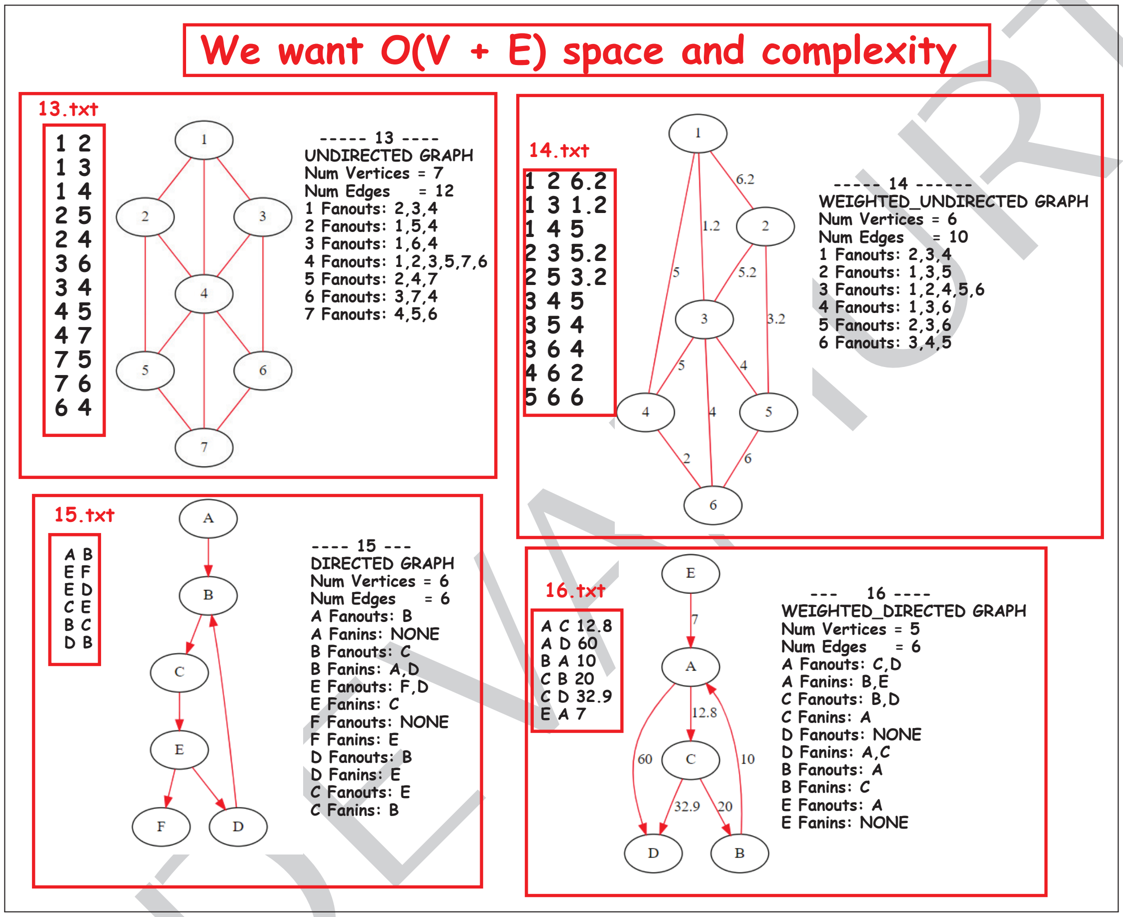
16.txt

```
A C 12.8
A D 60
B A 10
C B 20
C D 32.9
E A 7
```

```
WEIGHTED_DIRECTED GRAPH
Num Vertices = 5
Num Edges    = 6
A Fanouts: C,D
A FanIns: B,E
C Fanouts: B,D
C FanIns: A
D Fanouts: NONE
D FanIns: A,C
B Fanouts: A
B FanIns: C
E Fanouts: A
E FanIns: NONE
```

Figure 1.14: Representation of a directed weighted graph

1.5 Graph representation using fanins and fanouts lists



```
1 2
1 3
1 4
2 5
2 4
3 6
3 4
4 5
4 7
7 5
7 6
6 4
```

```
----- 13 ----
UNDIRECTED GRAPH
Num Vertices = 7
Num Edges    = 12
1 Fanouts: 2,3,4
2 Fanouts: 1,5,4
3 Fanouts: 1,6,4
4 Fanouts: 1,2,3,5,7,6
5 Fanouts: 2,4,7
6 Fanouts: 3,7,4
7 Fanouts: 4,5,6
```

```
1 2 6.2
1 3 1.2
1 4 5
2 3 5.2
2 5 3.2
3 4 5
3 5 4
3 6 4
4 6 2
5 6 6
```

```
----- 14 ------
WEIGHTED_UNDIRECTED GRAPH
Num Vertices = 6
Num Edges    = 10
1 Fanouts: 2,3,4
2 Fanouts: 1,3,5
3 Fanouts: 1,2,4,5,6
4 Fanouts: 1,3,6
5 Fanouts: 2,3,6
6 Fanouts: 3,4,5
```

```
A B
E F
E D
C E
B C
D B
```

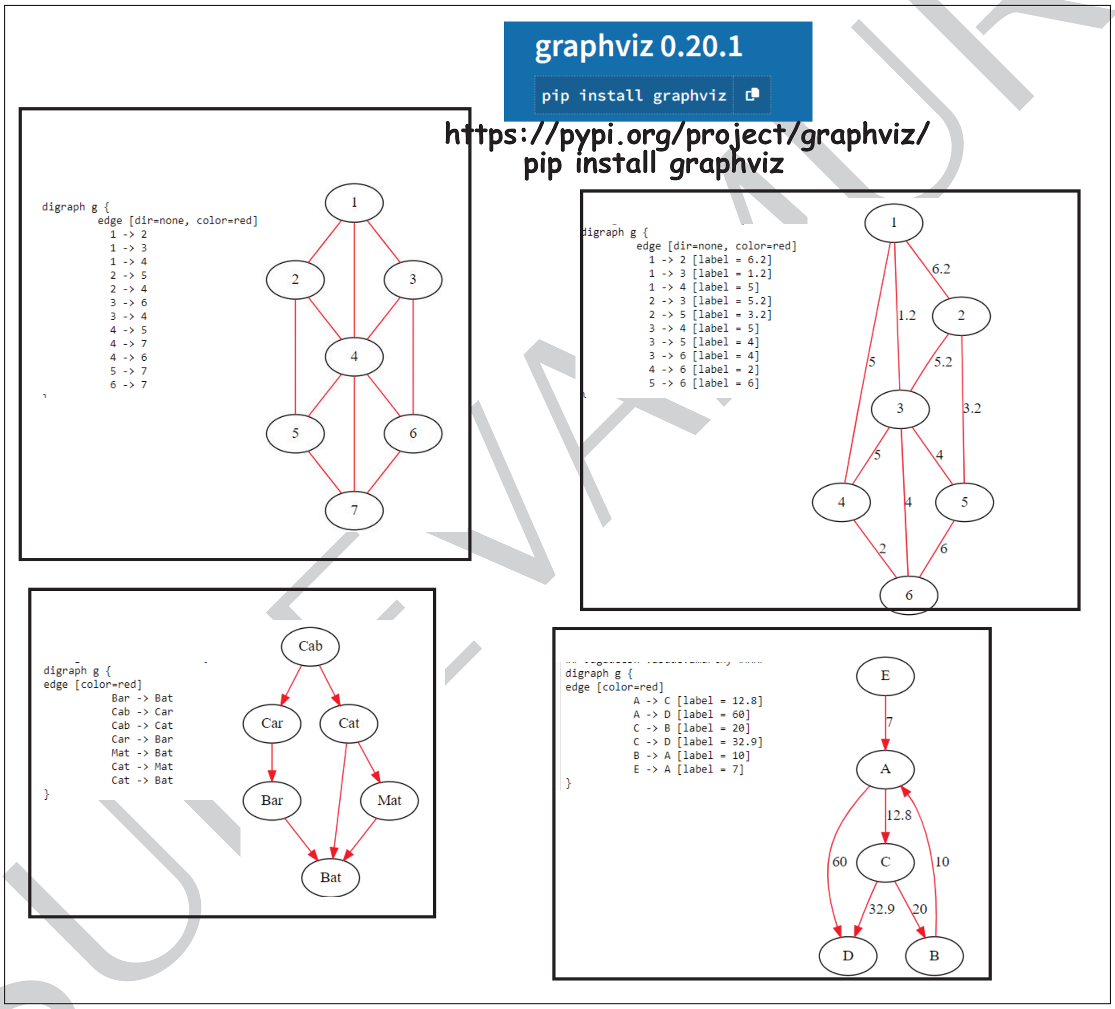
```
---- 15 ---
DIRECTED GRAPH
Num Vertices = 6
Num Edges    = 6
A Fanouts: B
A Fanins: NONE
B Fanouts: C
B Fanins: A,D
E Fanouts: F,D
E Fanins: C
F Fanouts: NONE
F Fanins: E
D Fanouts: B
D Fanins: E
C Fanouts: E
C Fanins: B
```

```
A C 12.8
A D 60
B A 10
C B 20
C D 32.9
E A 7
```

```
---    16 ----
WEIGHTED_DIRECTED GRAPH
Num Vertices = 5
Num Edges    = 6
A Fanouts: C,D
A Fanins: B,E
C Fanouts: B,D
C Fanins: A
D Fanouts: NONE
D Fanins: A,C
B Fanouts: A
B Fanins: C
E Fanouts: A
E Fanins: NONE
```

Figure 1.15: Representation of a graph

1.6 graphviz package



graphviz 0.20.1

```
pip install graphviz
```

https://pypi.org/project/graphviz/
pip install graphviz

```
digraph g {
        edge [dir=none, color=red]
          1 -> 2
          1 -> 3
          1 -> 4
          2 -> 5
          2 -> 4
          3 -> 6
          3 -> 4
          4 -> 5
          4 -> 7
          4 -> 6
          5 -> 7
          6 -> 7
}
```

```
digraph g {
        edge [dir=none, color=red]
          1 -> 2 [label = 6.2]
          1 -> 3 [label = 1.2]
          1 -> 4 [label = 5]
          2 -> 3 [label = 5.2]
          2 -> 5 [label = 3.2]
          3 -> 4 [label = 5]
          3 -> 5 [label = 4]
          3 -> 6 [label = 4]
          4 -> 6 [label = 2]
          5 -> 6 [label = 6]
}
```

```
digraph g {
edge [color=red]
          Bar -> Bat
          Cab -> Car
          Cab -> Cat
          Car -> Bar
          Mat -> Bat
          Cat -> Mat
          Cat -> Bat
}
```

```
digraph g {
edge [color=red]
          A -> C [label = 12.8]
          A -> D [label = 60]
          C -> B [label = 20]
          C -> D [label = 32.9]
          B -> A [label = 10]
          E -> A [label = 7]
}
```

Figure 1.16: *grapviz* pacakge

Drawing graph

Copyright: Jagadeesh Vasudevamurthy

filename: graphviz.ipynb

Basic imports

In [42]:
```
import sys
import os
from graphviz import Digraph
print(sys.version)
```

```
3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
```

Generic Read dot file

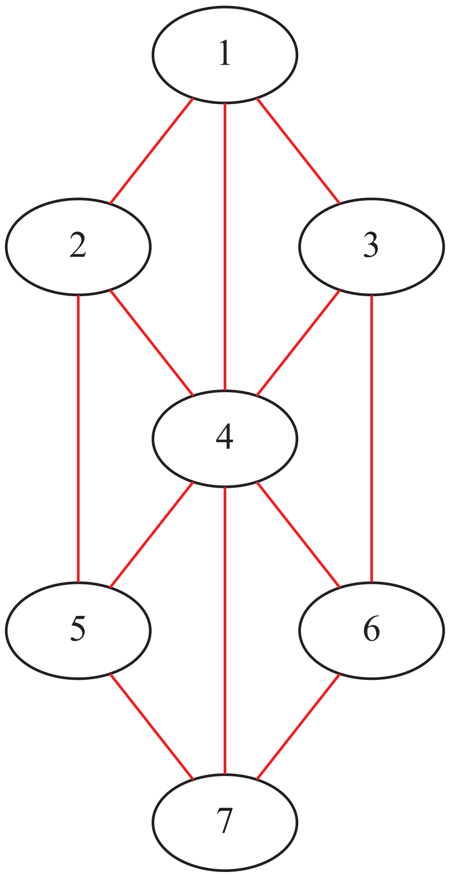
In [43]:
```
from graphviz import Source
def readDotFile(filename:'string')->'dot_graph':
    Base = "C:\\Users\\jag\\OneDrive\\vasu\\work\\py3\\objects\\py3\\py3\\
    file = Base + filename + ".dot"
    print(file)
    with open(file) as f:
        dot_graph = f.read()
    print(dot_graph)
    return dot_graph

```

Undirected Graphs With NO Weight

In [44]:
```
'''
#File: 13.dot
#Jagadeesh Vasudevamurthy
digraph g {
     edge [dir=none, color=red]
       1 -> 2
       1 -> 3
       1 -> 4
       2 -> 5
       2 -> 4
       3 -> 6
       3 -> 4
       4 -> 5
       4 -> 7
       4 -> 6
       5 -> 7
       6 -> 7
}
'''
Source(readDotFile("13"))
```

```
C:\Users\jag\OneDrive\vasu\work\py3\objects\py3\py3\dot\13.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
         edge [dir=none, color=red]
           1 -> 2
           1 -> 3
           1 -> 4
           2 -> 5
           2 -> 4
           3 -> 6
           3 -> 4
           4 -> 5
           4 -> 7
           4 -> 6
           5 -> 7
           6 -> 7
}
```

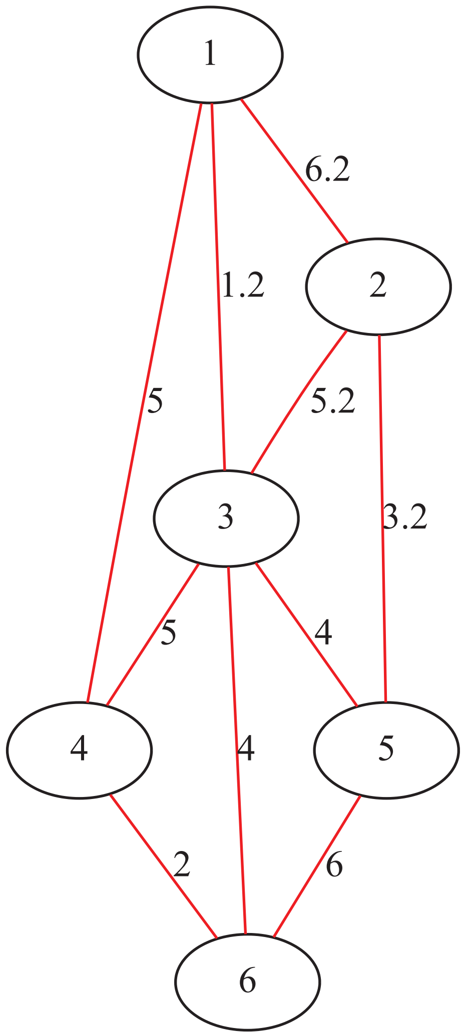
Out[44]:



In [45]:
```
'''
#FILE: 14.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
     edge [dir=none, color=red]
       1 -> 2 [label = 6.2]
       1 -> 3 [label = 1.2]
       1 -> 4 [label = 5]
       2 -> 3 [label = 5.2]
       2 -> 5 [label = 3.2]
       3 -> 4 [label = 5]
       3 -> 5 [label = 4]
       3 -> 6 [label = 4]
       4 -> 6 [label = 2]
       5 -> 6 [label = 6]
}
'''
Source(readDotFile("14"))
```

```
C:\Users\jag\OneDrive\vasu\work\py3\objects\py3\py3\dot\14.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
         edge [dir=none, color=red]
           1 -> 2 [label = 6.2]
           1 -> 3 [label = 1.2]
           1 -> 4 [label = 5]
           2 -> 3 [label = 5.2]
           2 -> 5 [label = 3.2]
           3 -> 4 [label = 5]
           3 -> 5 [label = 4]
           3 -> 6 [label = 4]
           4 -> 6 [label = 2]
           5 -> 6 [label = 6]
}
```

Out[45]:



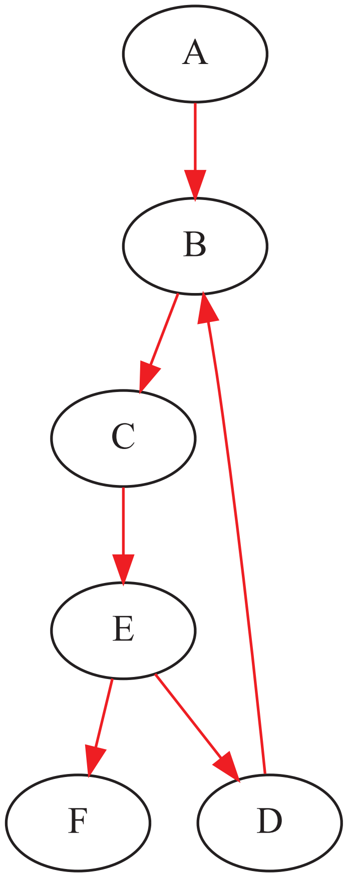
Directed Graphs With NO Weight

In [46]:

```
'''
#File: 15.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
edge [color=red]
        A -> B
        B -> C
        E -> F
        E -> D
        D -> B
        C -> E
}

'''
Source(readDotFile("15"))
```

```
C:\Users\jag\OneDrive\vasu\work\py3\objects\py3\py3\dot\15.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
edge [color=red]
          A -> B
          B -> C
          E -> F
          E -> D
          D -> B
          C -> E
}
```

Out[46]:

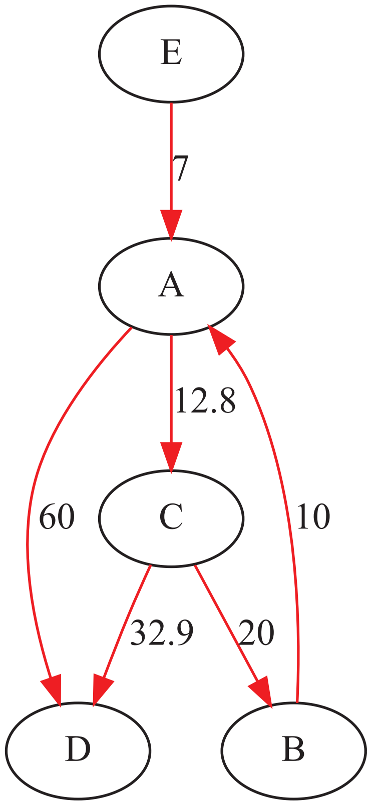


Directed Graph With Weight

In [47]:
```
'''
#File: 16,dot
## Jagadeesh Vasudevamurthy ####
digraph g {
edge [color=red]
        A -> C [label = 12.8]
        A -> D [label = 60]
        C -> B [label = 20]
        C -> D [label = 32.9]
        B -> A [label = 10]
        E -> A [label = 7]
}

'''
Source(readDotFile("16"))
```

```
C:\Users\jag\OneDrive\vasu\work\py3\objects\py3\py3\dot\16.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
edge [color=red]
          A -> C [label = 12.8]
          A -> D [label = 60]
          C -> B [label = 20]
          C -> D [label = 32.9]
          B -> A [label = 10]
          E -> A [label = 7]
}
```

Out[47]:

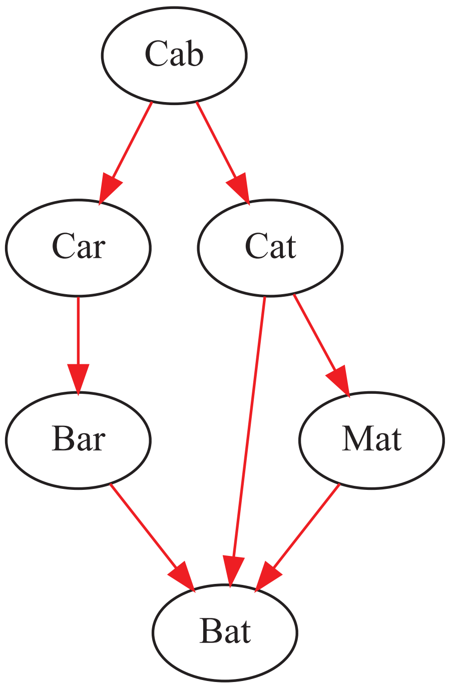


Directed Acyclic Graph (DAG)

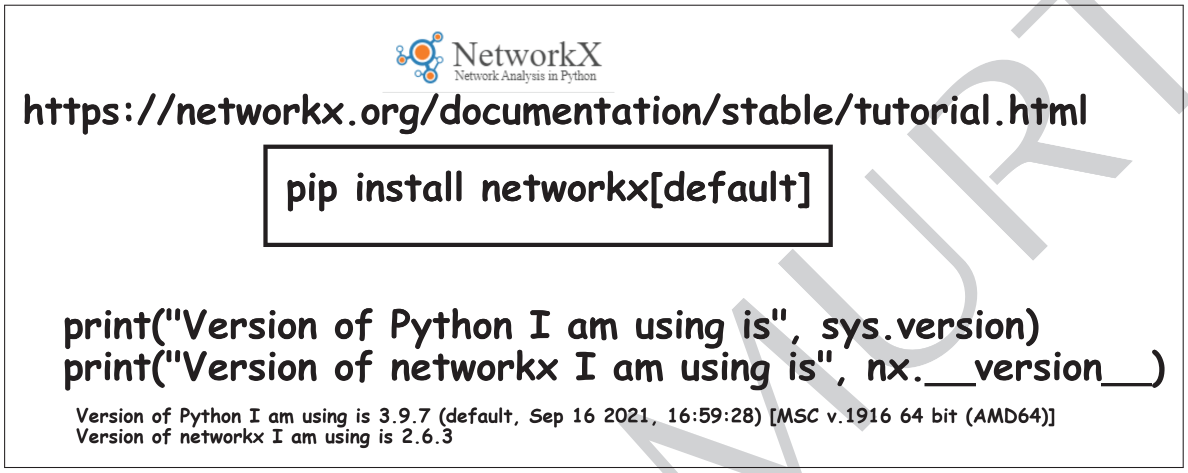
In [48]:
```
'''
#File: cat.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
edge [color=red]
        Bar -> Bat
        Cab -> Car
        Cab -> Cat
        Car -> Bar
        Mat -> Bat
        Cat -> Mat
        Cat -> Bat
}
'''
Source(readDotFile("cat"))
```

```
C:\Users\jag\OneDrive\vasu\work\py3\objects\py3\py3\dot\cat.dot
## Jagadeesh Vasudevamurthy ####
digraph g {
edge [color=red]
           Bar -> Bat
           Cab -> Car
           Cab -> Cat
           Car -> Bar
           Mat -> Bat
           Cat -> Mat
           Cat -> Bat
}
```

Out[48]:



1.7 networkx package


NetworkX
Network Analysis in Python

https://networkx.org/documentation/stable/tutorial.html

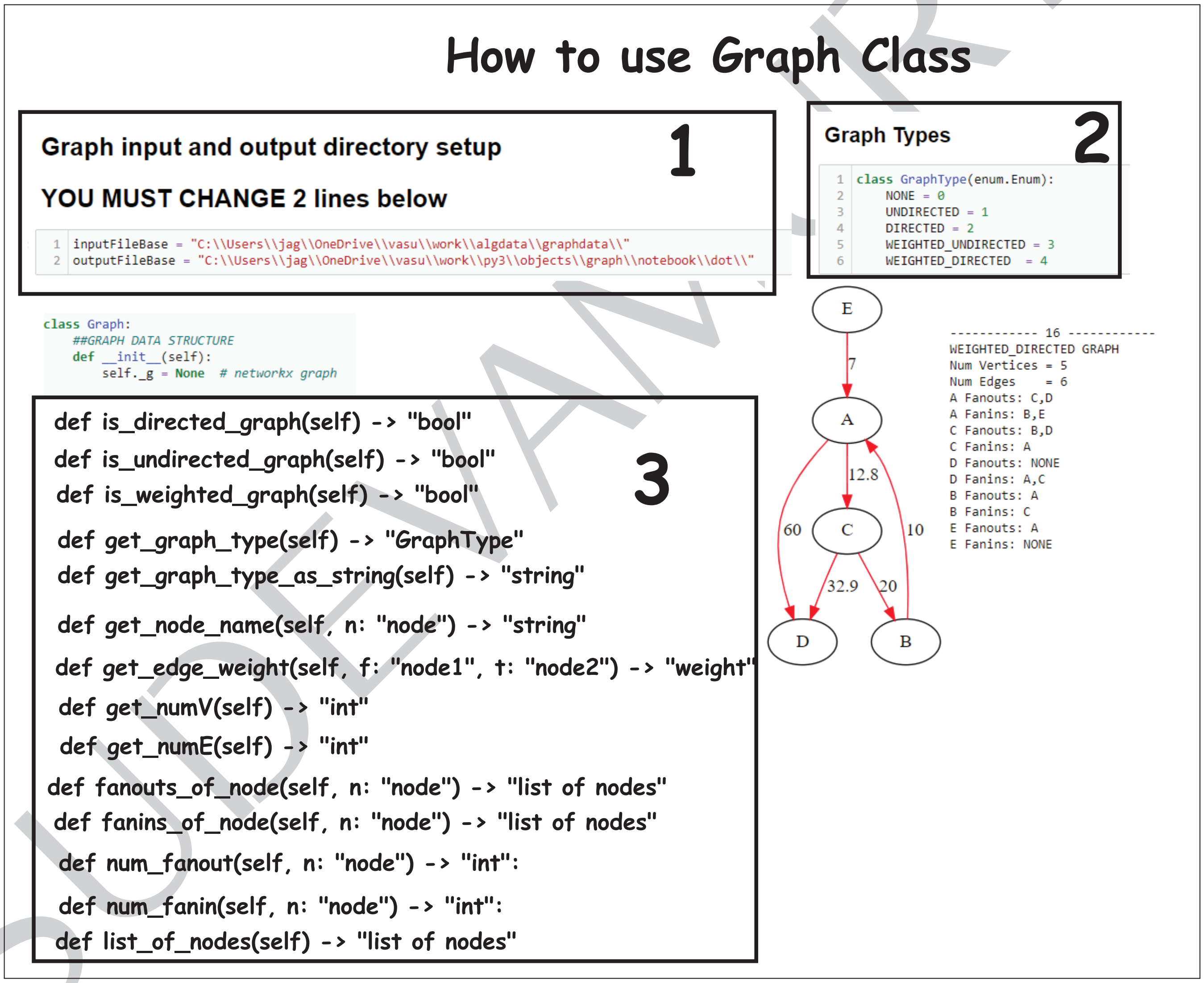
```
pip install networkx[default]
```

```
print("Version of Python I am using is", sys.version)
print("Version of networkx I am using is", nx.__version__)
```

```
Version of Python I am using is 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Version of networkx I am using is 2.6.3
```

Figure 1.17: *networkx* pacakge

1.8 class Graph



How to use Graph Class

Graph input and output directory setup 1

YOU MUST CHANGE 2 lines below

```
inputFileBase = "C:\\Users\\jag\\OneDrive\\vasu\\work\\algdata\\graphdata\\"
outputFileBase = "C:\\Users\\jag\\OneDrive\\vasu\\work\\py3\\objects\\graph\\notebook\\dot\\"
```

Graph Types 2

```
class GraphType(enum.Enum):
    NONE = 0
    UNDIRECTED = 1
    DIRECTED = 2
    WEIGHTED_UNDIRECTED = 3
    WEIGHTED_DIRECTED  = 4
```

```
class Graph:
    ##GRAPH DATA STRUCTURE
    def __init__(self):
        self._g = None  # networkx graph
```

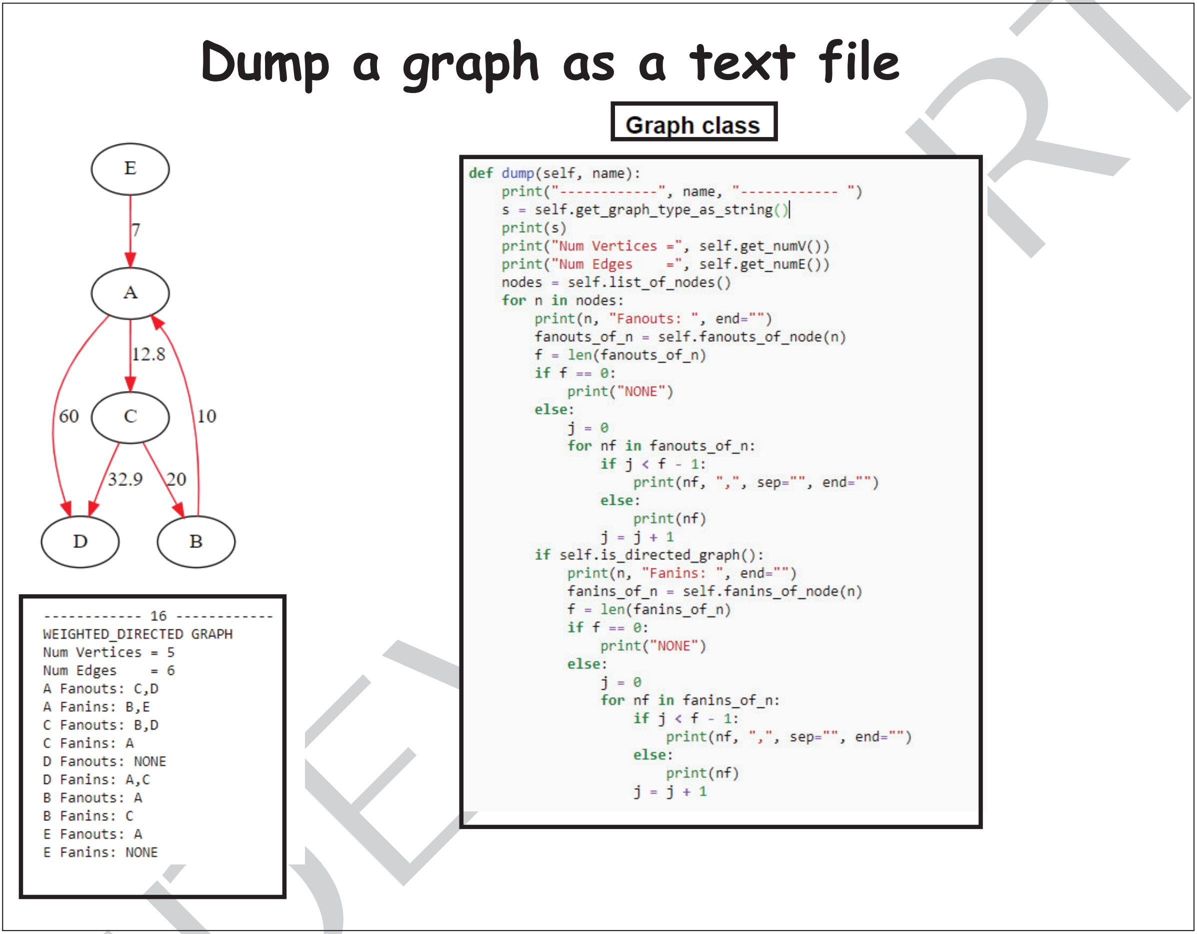
3

```
def is_directed_graph(self) -> "bool"
def is_undirected_graph(self) -> "bool"
def is_weighted_graph(self) -> "bool"
def get_graph_type(self) -> "GraphType"
def get_graph_type_as_string(self) -> "string"
def get_node_name(self, n: "node") -> "string"
def get_edge_weight(self, f: "node1", t: "node2") -> "weight"
def get_numV(self) -> "int"
def get_numE(self) -> "int"
def fanouts_of_node(self, n: "node") -> "list of nodes"
def fanins_of_node(self, n: "node") -> "list of nodes"
def num_fanout(self, n: "node") -> "int":
def num_fanin(self, n: "node") -> "int":
def list_of_nodes(self) -> "list of nodes"
```

```
------------ 16 ------------
WEIGHTED_DIRECTED GRAPH
Num Vertices = 5
Num Edges    = 6
A Fanouts: C,D
A Fanins: B,E
C Fanouts: B,D
C Fanins: A
D Fanouts: NONE
D Fanins: A,C
B Fanouts: A
B Fanins: C
E Fanouts: A
E Fanins: NONE
```

Figure 1.18: Graph public functions

1.9 Dump a graph as a text file

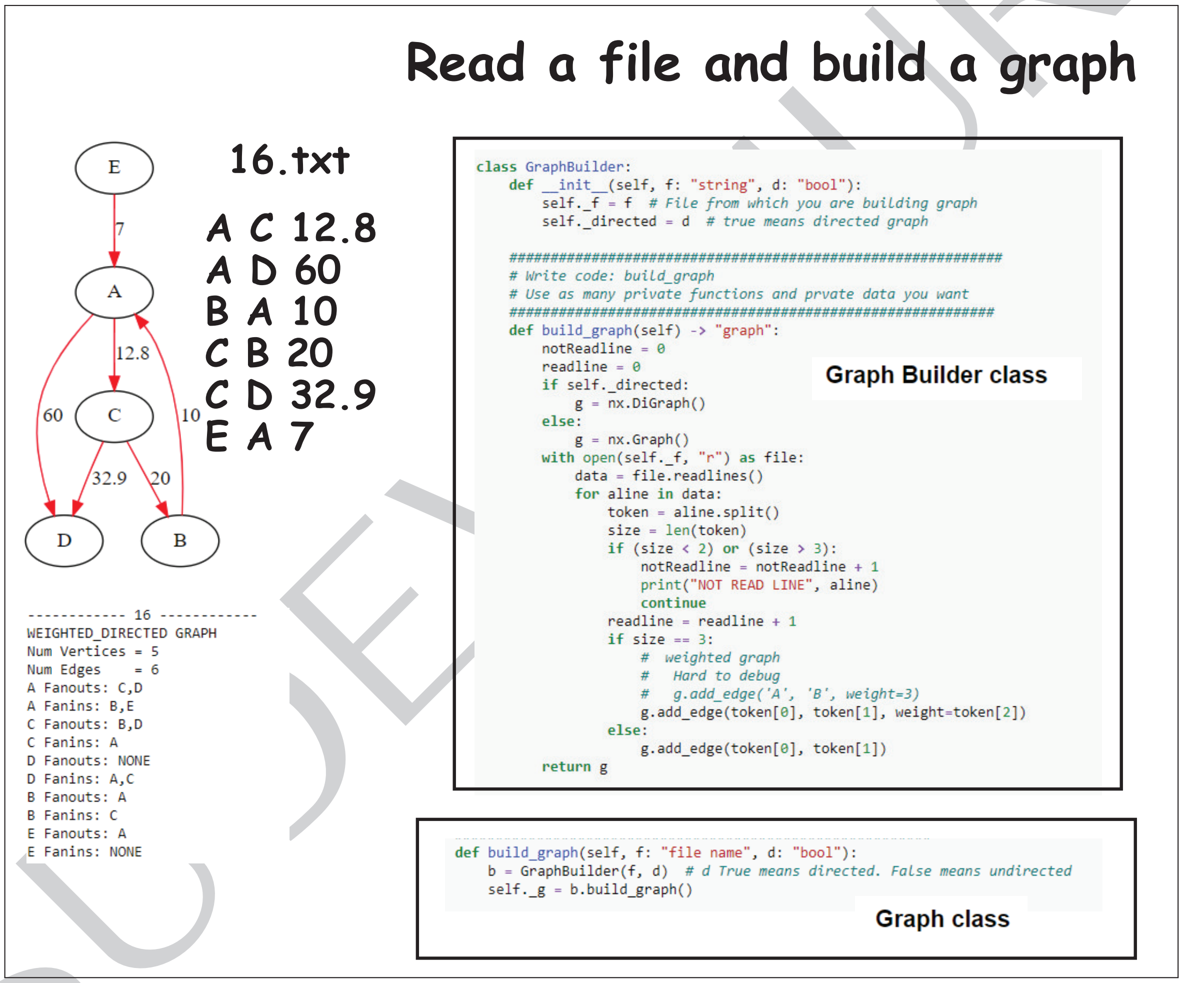


```
------------ 16 ------------
WEIGHTED_DIRECTED GRAPH
Num Vertices = 5
Num Edges    = 6
A Fanouts: C,D
A Fanins: B,E
C Fanouts: B,D
C Fanins: A
D Fanouts: NONE
D Fanins: A,C
B Fanouts: A
B Fanins: C
E Fanouts: A
E Fanins: NONE
```

```python
def dump(self, name):
    print("------------", name, "------------ ")
    s = self.get_graph_type_as_string()
    print(s)
    print("Num Vertices =", self.get_numV())
    print("Num Edges    =", self.get_numE())
    nodes = self.list_of_nodes()
    for n in nodes:
        print(n, "Fanouts: ", end="")
        fanouts_of_n = self.fanouts_of_node(n)
        f = len(fanouts_of_n)
        if f == 0:
            print("NONE")
        else:
            j = 0
            for nf in fanouts_of_n:
                if j < f - 1:
                    print(nf, ",", sep="", end="")
                else:
                    print(nf)
                j = j + 1
        if self.is_directed_graph():
            print(n, "Fanins: ", end="")
            fanins_of_n = self.fanins_of_node(n)
            f = len(fanins_of_n)
            if f == 0:
                print("NONE")
            else:
                j = 0
                for nf in fanins_of_n:
                    if j < f - 1:
                        print(nf, ",", sep="", end="")
                    else:
                        print(nf)
                    j = j + 1
```

Figure 1.19: Dump a graph as a text file

1.10 Build a graph from a file



Read a file and build a graph

16.txt

```
A C 12.8
A D 60
B A 10
C B 20
C D 32.9
E A 7
```

```
------------ 16 ------------
WEIGHTED_DIRECTED GRAPH
Num Vertices = 5
Num Edges    = 6
A Fanouts: C,D
A Fanins: B,E
C Fanouts: B,D
C Fanins: A
D Fanouts: NONE
D Fanins: A,C
B Fanouts: A
B Fanins: C
E Fanouts: A
E Fanins: NONE
```

```python
class GraphBuilder:
    def __init__(self, f: "string", d: "bool"):
        self._f = f  # File from which you are building graph
        self._directed = d  # true means directed graph

    ############################################################
    # Write code: build_graph
    # Use as many private functions and prvate data you want
    ###########################################################
    def build_graph(self) -> "graph":
        notReadline = 0
        readline = 0
        if self._directed:
            g = nx.DiGraph()
        else:
            g = nx.Graph()
        with open(self._f, "r") as file:
            data = file.readlines()
            for aline in data:
                token = aline.split()
                size = len(token)
                if (size < 2) or (size > 3):
                    notReadline = notReadline + 1
                    print("NOT READ LINE", aline)
                    continue
                readline = readline + 1
                if size == 3:
                    #  weighted graph
                    #   Hard to debug
                    #   g.add_edge('A', 'B', weight=3)
                    g.add_edge(token[0], token[1], weight=token[2])
                else:
                    g.add_edge(token[0], token[1])
        return g
```

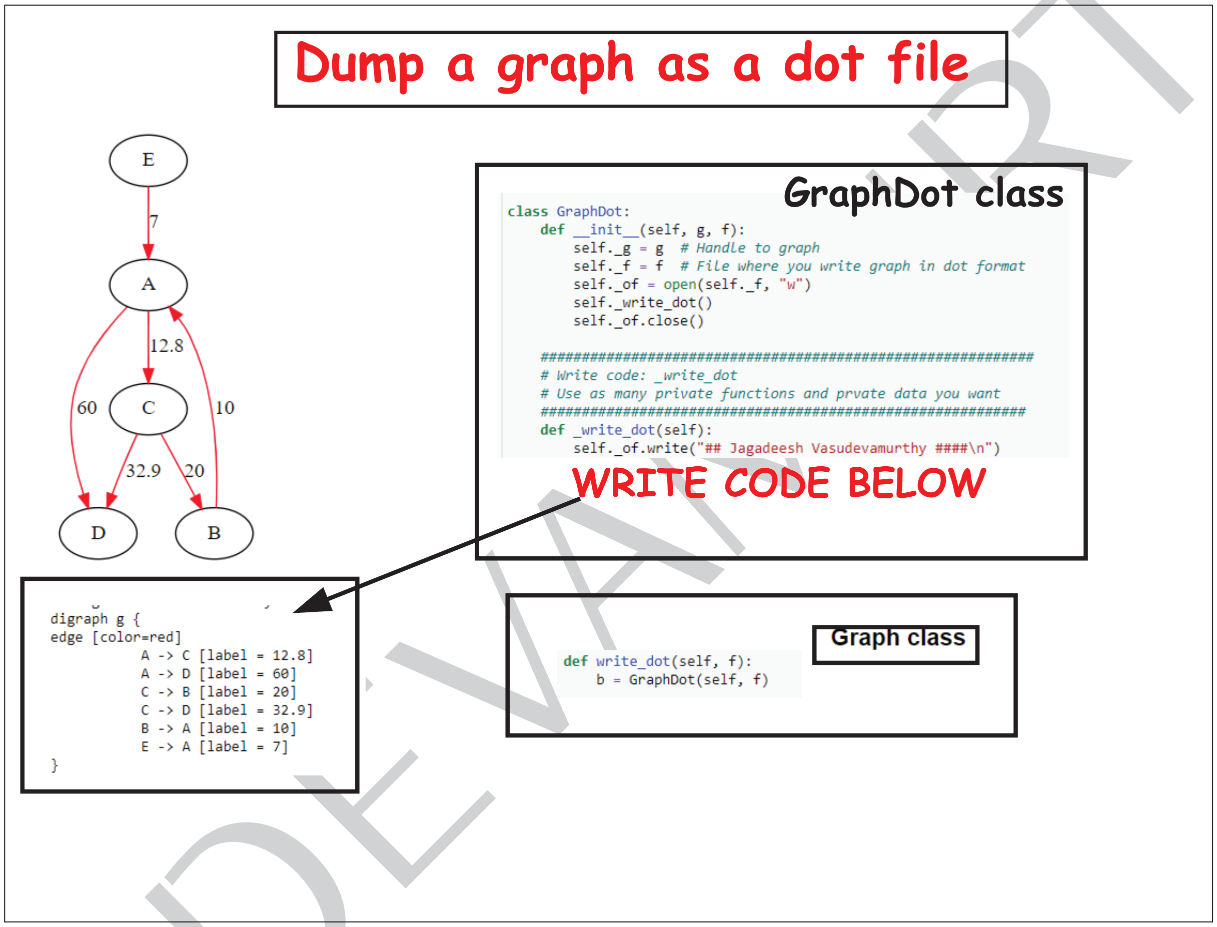
Graph Builder class

```python
def build_graph(self, f: "file name", d: "bool"):
    b = GraphBuilder(f, d)  # d True means directed. False means undirected
    self._g = b.build_graph()
```

Graph class

Figure 1.20: **class GraphBuilder**

1.11 Write a graph as a dot file



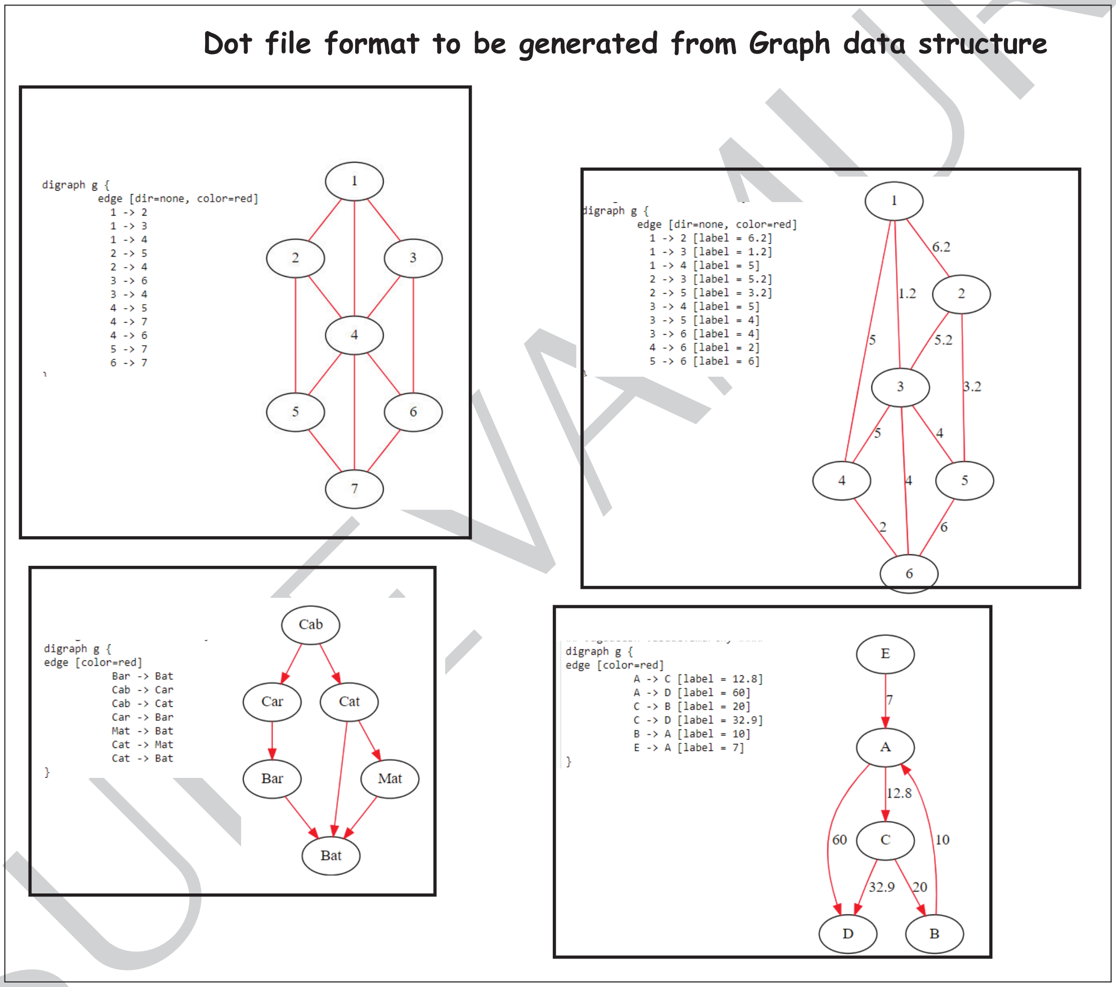
```
class GraphDot:
    def __init__(self, g, f):
        self._g = g  # Handle to graph
        self._f = f  # File where you write graph in dot format
        self._of = open(self._f, "w")
        self._write_dot()
        self._of.close()

    ###############################################################
    # Write code: _write_dot
    # Use as many private functions and prvate data you want
    ###############################################################
    def _write_dot(self):
        self._of.write("## Jagadeesh Vasudevamurthy ####\n")
```

```
digraph g {
edge [color=red]
            A -> C [label = 12.8]
            A -> D [label = 60]
            C -> B [label = 20]
            C -> D [label = 32.9]
            B -> A [label = 10]
            E -> A [label = 7]
}
```

```
def write_dot(self, f):
    b = GraphDot(self, f)
```

Figure 1.21: class GraphDot

1.11.1 Various dot file examples



```
digraph g {
          edge [dir=none, color=red]
            1 -> 2
            1 -> 3
            1 -> 4
            2 -> 5
            2 -> 4
            3 -> 6
            3 -> 4
            4 -> 5
            4 -> 7
            4 -> 6
            5 -> 7
            6 -> 7
}
```

```
digraph g {
          edge [dir=none, color=red]
            1 -> 2 [label = 6.2]
            1 -> 3 [label = 1.2]
            1 -> 4 [label = 5]
            2 -> 3 [label = 5.2]
            2 -> 5 [label = 3.2]
            3 -> 4 [label = 5]
            3 -> 5 [label = 4]
            3 -> 6 [label = 4]
            4 -> 6 [label = 2]
            5 -> 6 [label = 6]
}
```

```
digraph g {
edge [color=red]
            Bar -> Bat
            Cab -> Car
            Cab -> Cat
            Car -> Bar
            Mat -> Bat
            Cat -> Mat
            Cat -> Bat
}
```

```
digraph g {
edge [color=red]
            A -> C [label = 12.8]
            A -> D [label = 60]
            C -> B [label = 20]
            C -> D [label = 32.9]
            B -> A [label = 10]
            E -> A [label = 7]
}
```

Figure 1.22: Various dot file examples

1.12 Loops in a graph

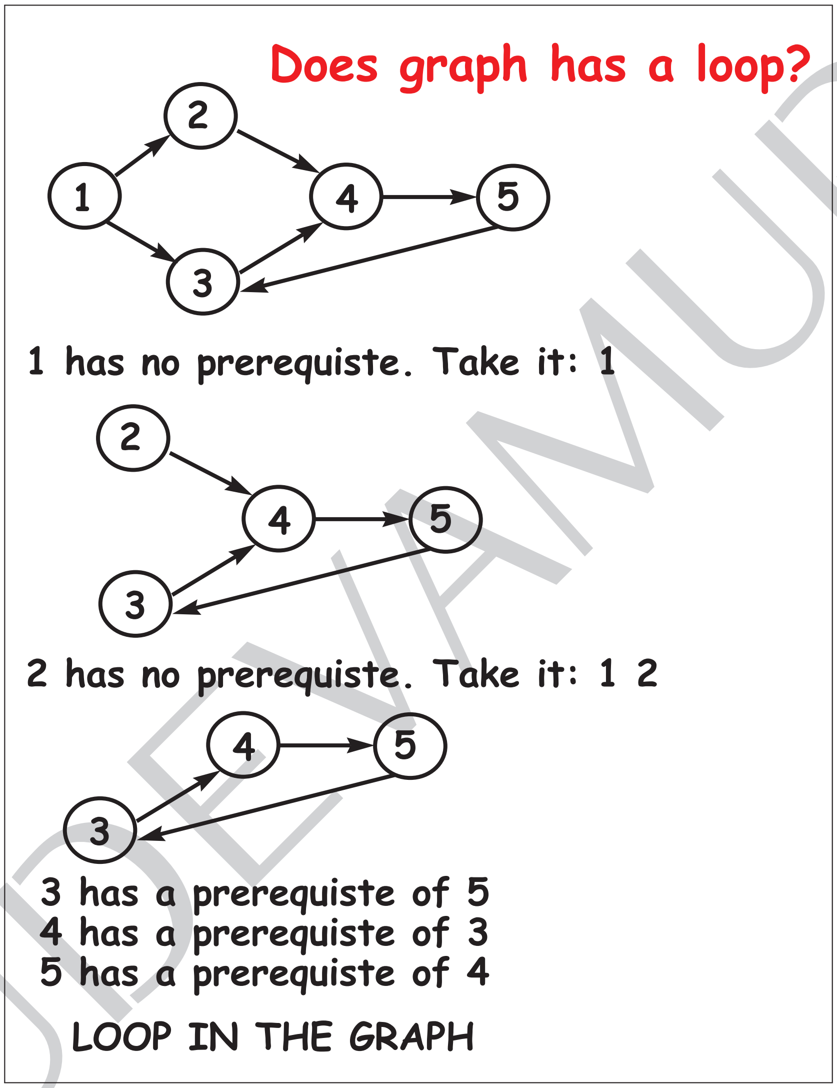
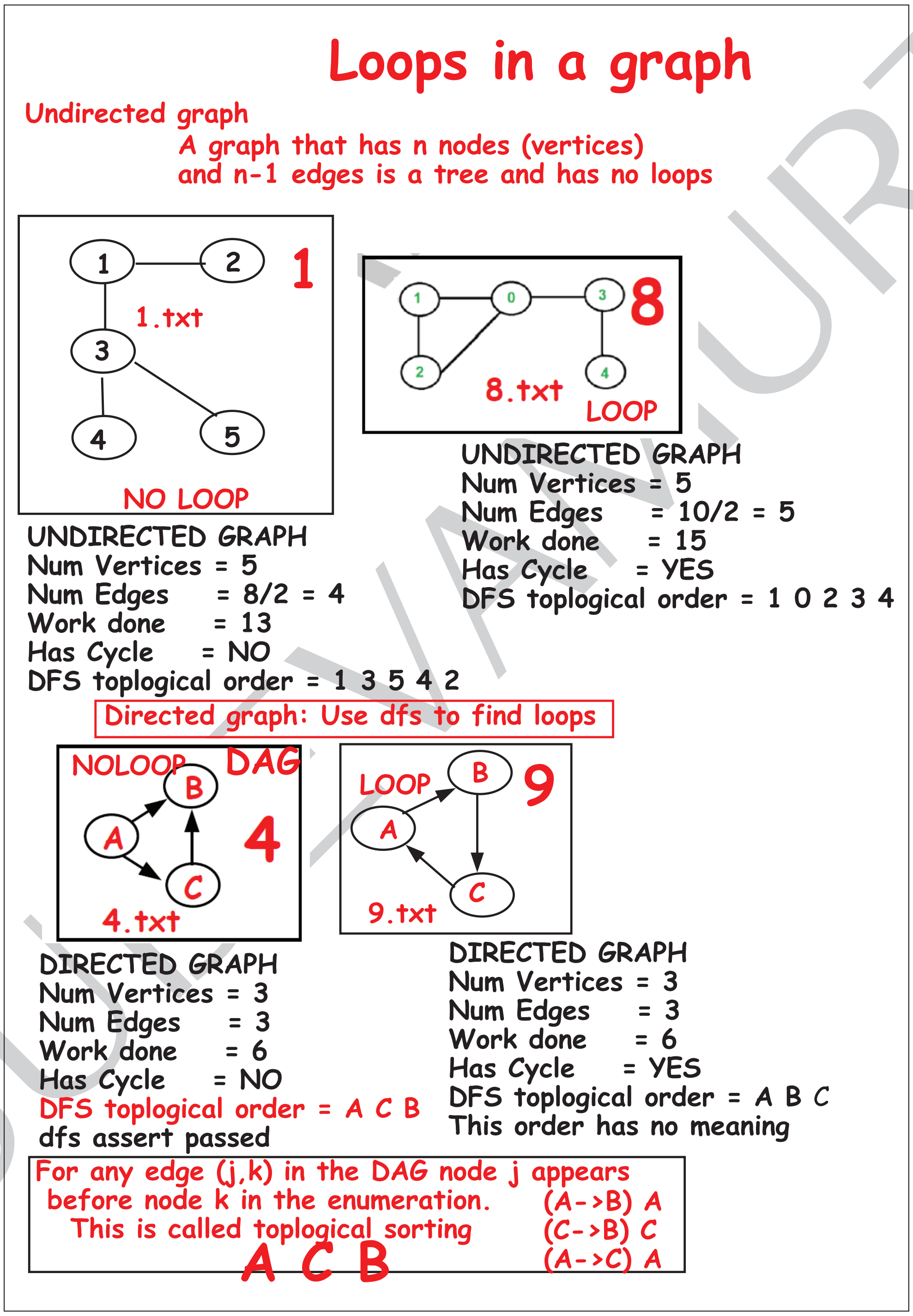


Figure 1.23: Completing courses in an university



Loops in a graph

Undirected graph

A graph that has n nodes (vertices) and n-1 edges is a tree and has no loops

UNDIRECTED GRAPH
Num Vertices = 5
Num Edges = 8/2 = 4
Work done = 13
Has Cycle = NO
DFS toplogical order = 1 3 5 4 2

UNDIRECTED GRAPH
Num Vertices = 5
Num Edges = 10/2 = 5
Work done = 15
Has Cycle = YES
DFS toplogical order = 1 0 2 3 4

Directed graph: Use dfs to find loops

DIRECTED GRAPH
Num Vertices = 3
Num Edges = 3
Work done = 6
Has Cycle = NO
DFS toplogical order = A C B
dfs assert passed

DIRECTED GRAPH
Num Vertices = 3
Num Edges = 3
Work done = 6
Has Cycle = YES
DFS toplogical order = A B C
This order has no meaning

For any edge (j,k) in the DAG node j appears before node k in the enumeration. This is called toplogical sorting

(A->B) A
(C->B) C
(A->C) A

A C B

Figure 1.24: Loop in a graph

1.13 Depth first search using time stamps

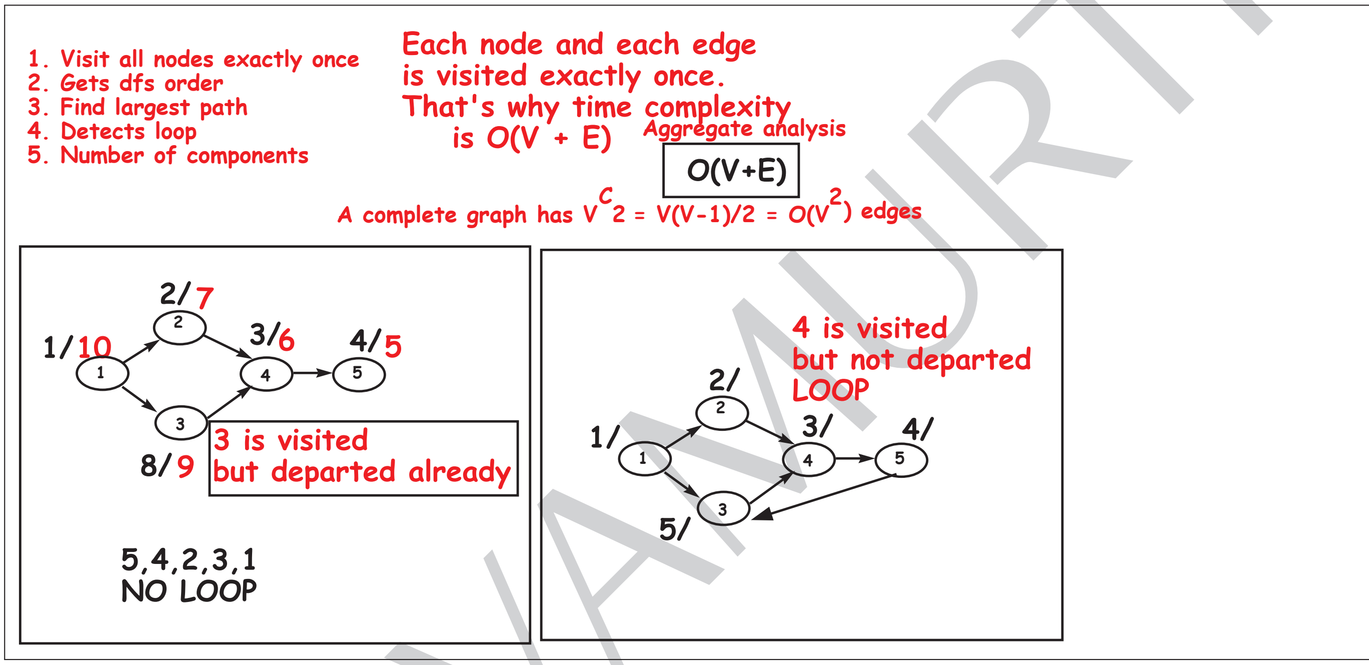

1. Visit all nodes exactly once
2. Gets dfs order
3. Find largest path
4. Detects loop
5. Number of components
Each node and each edge is visited exactly once.
That's why time complexity is O(V + E)
Aggregate analysis
O(V+E)
A complete graph has V C 2 = V(V-1)/2 = O(V2) edges
1/10
2/7
3/6
4/5
8/9
3 is visited but departed already
5,4,2,3,1
NO LOOP
4 is visited but not departed
LOOP
1/
2/
3/
4/
5/

Figure 1.25: Depth first search on a directed graph using time stamps

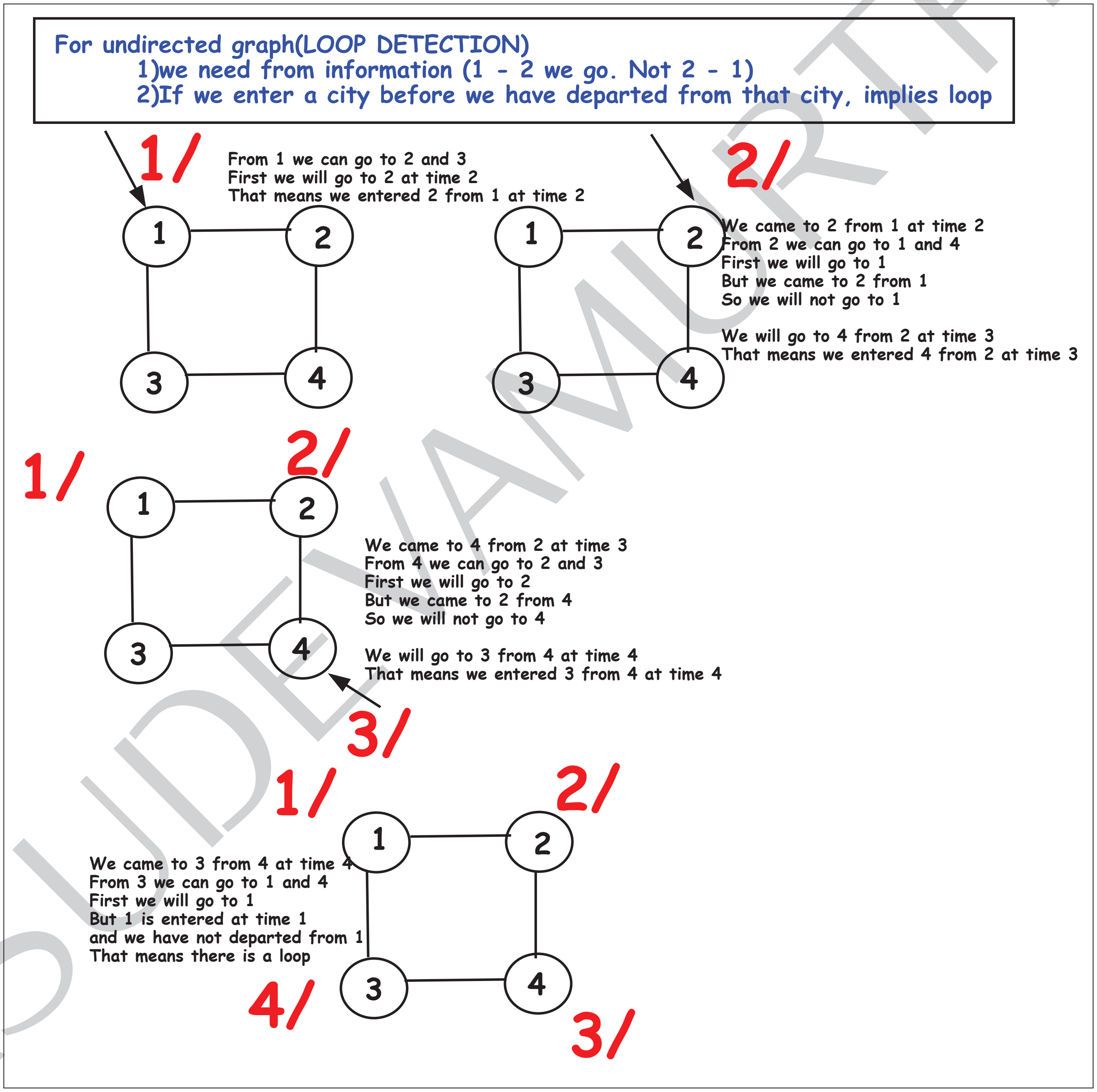
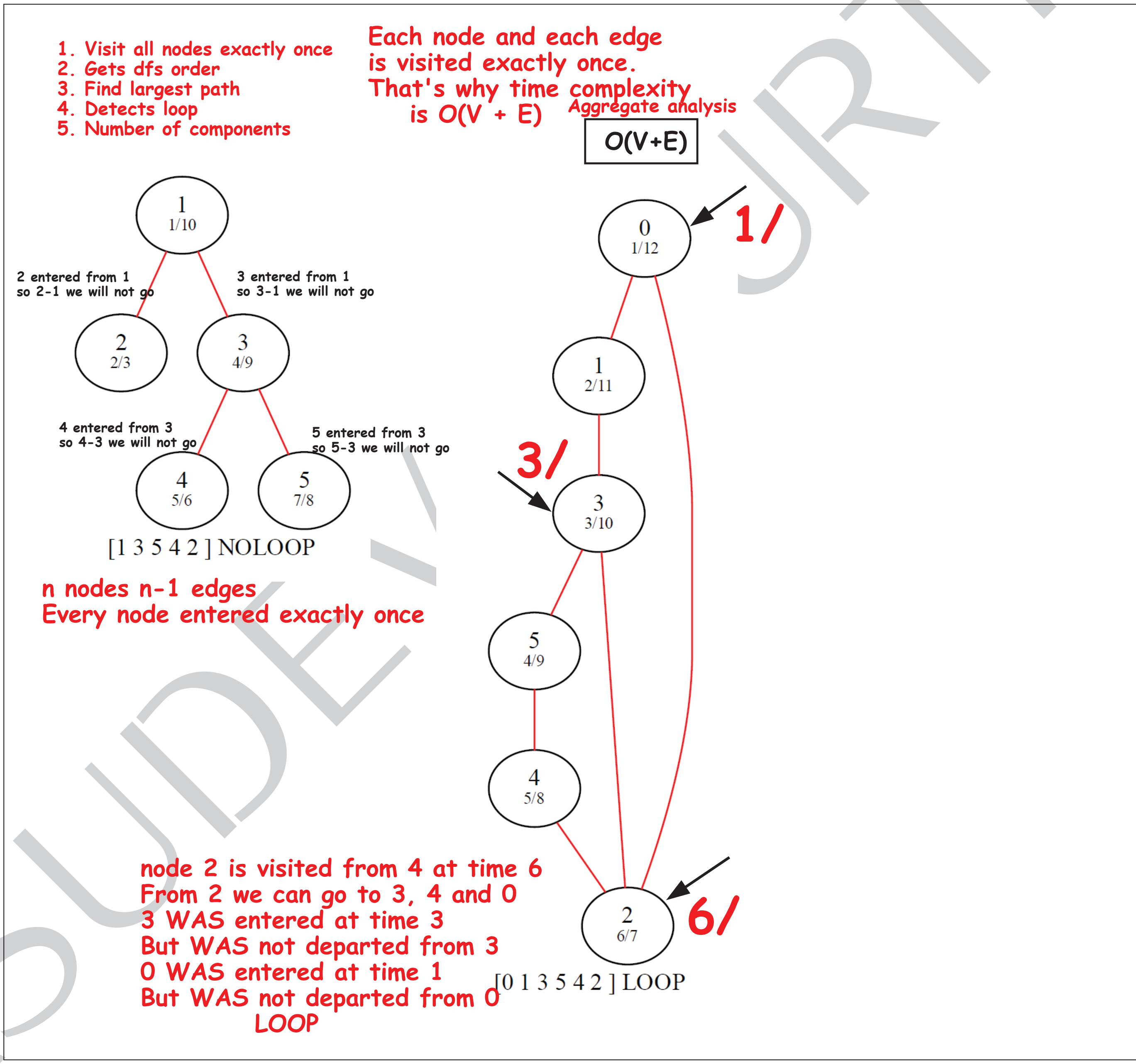

For undirected graph(LOOP DETECTION)
1)we need from information (1 - 2 we go. Not 2 - 1)
2)If we enter a city before we have departed from that city, implies loop
1/
From 1 we can go to 2 and 3
First we will go to 2 at time 2
That means we entered 2 from 1 at time 2
1
2
3
4
2/
We came to 2 from 1 at time 2
From 2 we can go to 1 and 4
First we will go to 1
But we came to 2 from 1
So we will not go to 1
We will go to 4 from 2 at time 3
That means we entered 4 from 2 at time 3
1/
2/
We came to 4 from 2 at time 3
From 4 we can go to 2 and 3
First we will go to 2
But we came to 2 from 4
So we will not go to 4
We will go to 3 from 4 at time 4
That means we entered 3 from 4 at time 4
3/
1/
2/
We came to 3 from 4 at time 4
From 3 we can go to 1 and 4
First we will go to 1
But 1 is entered at time 1
and we have not departed from 1
That means there is a loop
4/
3/

Figure 1.26: Depth first search on an undirected graph using time stamps and **from**


1. Visit all nodes exactly once
2. Gets dfs order
3. Find largest path
4. Detects loop
5. Number of components

Each node and each edge is visited exactly once.
That's why time complexity is O(V + E)

Aggregate analysis

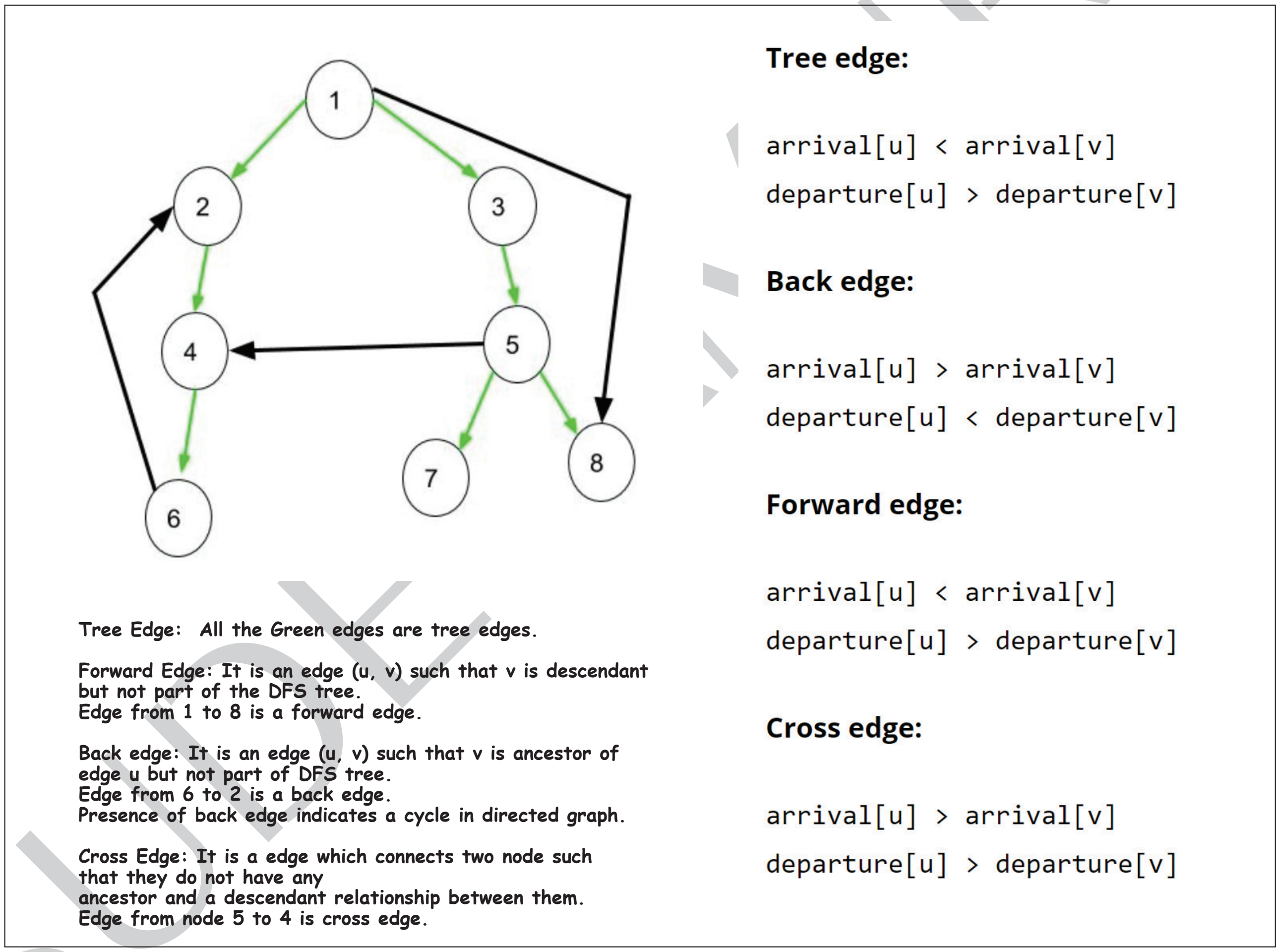
$\boxed{O(V+E)}$

n nodes n-1 edges
Every node entered exactly once

node 2 is visited from 4 at time 6
From 2 we can go to 3, 4 and 0
3 WAS entered at time 3
But WAS not departed from 3
0 WAS entered at time 1
But WAS not departed from 0
LOOP

1 1/10
2 entered from 1 so 2-1 we will not go
3 entered from 1 so 3-1 we will not go
2 2/3
3 4/9
4 entered from 3 so 4-3 we will not go
5 entered from 3 so 5-3 we will not go
4 5/6
5 7/8
[1 3 5 4 2] NOLOOP
0 1/12
1/
1 2/11
3/
3 3/10
5 4/9
4 5/8
2 6/7
6/
[0 1 3 5 4 2] LOOP

Figure 1.27: Depth first search on an undirected graph



Tree Edge: All the Green edges are tree edges.

Forward Edge: It is an edge (u, v) such that v is descendant but not part of the DFS tree.
Edge from 1 to 8 is a forward edge.

Back edge: It is an edge (u, v) such that v is ancestor of edge u but not part of DFS tree.
Edge from 6 to 2 is a back edge.
Presence of back edge indicates a cycle in directed graph.

Cross Edge: It is a edge which connects two node such that they do not have any
ancestor and a descendant relationship between them.
Edge from node 5 to 4 is cross edge.

Tree edge:

```
arrival[u] < arrival[v]
departure[u] > departure[v]
```

Back edge:

```
arrival[u] > arrival[v]
departure[u] < departure[v]
```

Forward edge:

```
arrival[u] < arrival[v]
departure[u] > departure[v]
```

Cross edge:

```
arrival[u] > arrival[v]
departure[u] > departure[v]
```

Figure 1.28: classification of edges in a directed graph

1.13.1 Depth first search on a undirected graph with no loop

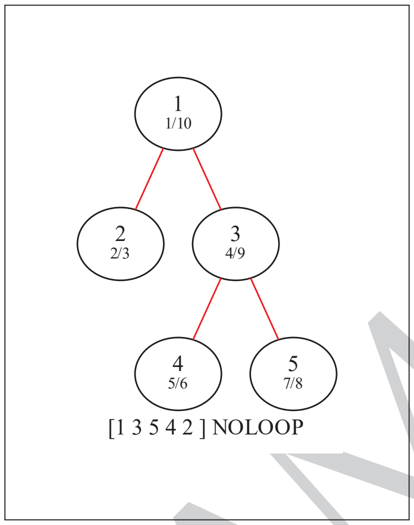


Figure 1.29: undirected graph with no loop

1.13.2 Depth first search on a undirected graph with loop

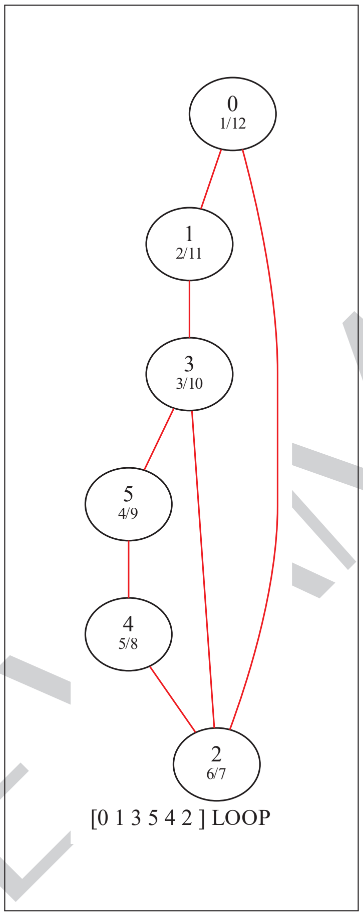


Figure 1.30: undirected graph with loop

```
## Jagadeesh Vasudevamurthy ####
## dot -Tpdf C:\scratch\outputs\dot\udf1dfs.dot -o C:\scratch\outputs\dot\udf1dfs.do
digraph g {
   label = "[0 1 3 5 4 2 ] LOOP"
   0[label = <0<BR /><FONT POINT-SIZE="10">1/12</FONT>>]
   1[label = <1<BR /><FONT POINT-SIZE="10">2/11</FONT>>]
   3[label = <3<BR /><FONT POINT-SIZE="10">3/10</FONT>>]
   5[label = <5<BR /><FONT POINT-SIZE="10">4/9</FONT>>]
   4[label = <4<BR /><FONT POINT-SIZE="10">5/8</FONT>>]
   2[label = <2<BR /><FONT POINT-SIZE="10">6/7</FONT>>]
edge [dir=none, color=red]
   0 -> 1
   0 -> 2
   1 -> 3
   3 -> 5
   3 -> 2
   5 -> 4
   4 -> 2
}
```

1.13.3 Depth first search on a directed graph with no loop

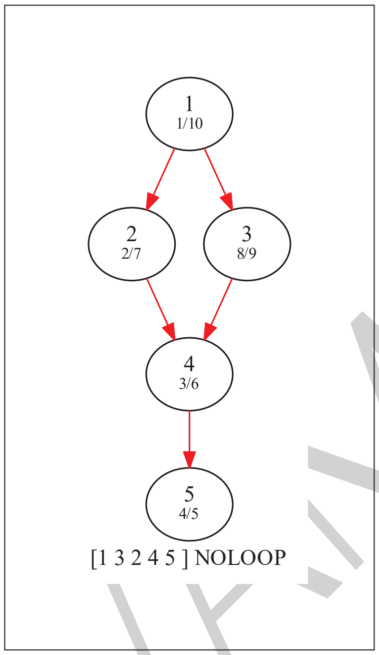


Figure 1.31: directed graph with no loop

```
## Jagadeesh Vasudevamurthy ####
## dot -Tpdf C:\scratch\outputs\dot\2dfs.dot -o C:\scratch\outputs\dot\2dfs.dot.pdf
digraph g {
   label = "[1 3 2 4 5 ] NOLOOP"
   1[label = <1<BR /><FONT POINT-SIZE="10">1/10</FONT>>]
   2[label = <2<BR /><FONT POINT-SIZE="10">2/7</FONT>>]
   3[label = <3<BR /><FONT POINT-SIZE="10">8/9</FONT>>]
   4[label = <4<BR /><FONT POINT-SIZE="10">3/6</FONT>>]
   5[label = <5<BR /><FONT POINT-SIZE="10">4/5</FONT>>]
edge [color=red]
   1 -> 2
   1 -> 3
   2 -> 4
   3 -> 4
   4 -> 5
}
```

1.13.4 Depth first search on a directed graph with loop

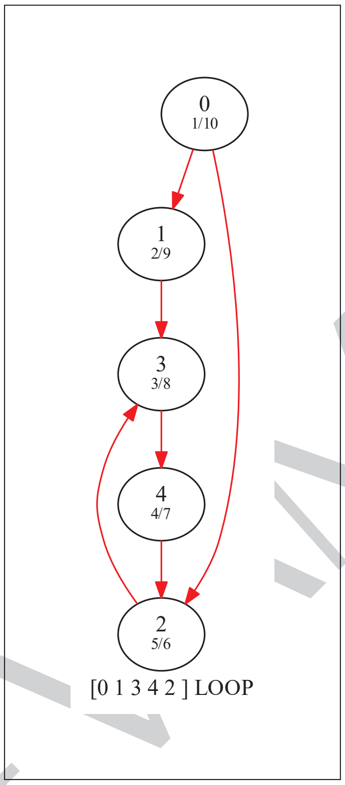


Figure 1.32: directed graph with loop

```
## Jagadeesh Vasudevamurthy ####
## dot -Tpdf C:\scratch\outputs\dot\3dfs.dot -o C:\scratch\outputs\dot\3dfs.dot.pdf
digraph g {
   label = "[0 1 3 4 2 ] LOOP"
   0[label = <0<BR /><FONT POINT-SIZE="10">1/10</FONT>>]
   1[label = <1<BR /><FONT POINT-SIZE="10">2/9</FONT>>]
   2[label = <2<BR /><FONT POINT-SIZE="10">5/6</FONT>>]
   3[label = <3<BR /><FONT POINT-SIZE="10">3/8</FONT>>]
   4[label = <4<BR /><FONT POINT-SIZE="10">4/7</FONT>>]
edge [color=red]
   0 -> 1
   0 -> 2
   1 -> 3
   2 -> 3
   3 -> 4
   4 -> 2
}
```

1.13.5 Depth first search on a directed graph with no loop

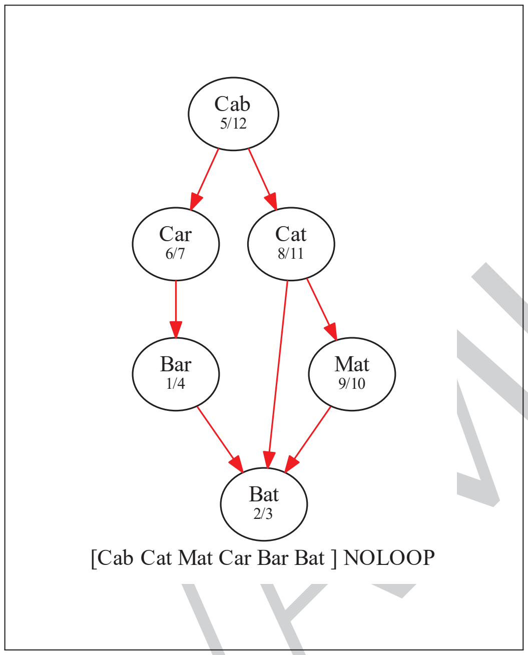


Figure 1.33: directed graph with no loop

```
## Jagadeesh Vasudevamurthy ####
## dot -Tpdf C:\scratch\outputs\dot\catdfs.dot -o C:\scratch\outputs\dot\catdfs.dot.
digraph g {
   label = "[Cab Cat Mat Car Bar Bat ] NOLOOP"
   Bar[label = <Bar<BR /><FONT POINT-SIZE="10">1/4</FONT>>]
   Bat[label = <Bat<BR /><FONT POINT-SIZE="10">2/3</FONT>>]
   Cab[label = <Cab<BR /><FONT POINT-SIZE="10">5/12</FONT>>]
   Car[label = <Car<BR /><FONT POINT-SIZE="10">6/7</FONT>>]
   Mat[label = <Mat<BR /><FONT POINT-SIZE="10">9/10</FONT>>]
   Cat[label = <Cat<BR /><FONT POINT-SIZE="10">8/11</FONT>>]
edge [color=red]
   Bar -> Bat
   Cab -> Car
   Cab -> Cat
   Car -> Bar
   Mat -> Bat
   Cat -> Bat
   Cat -> Mat
}
```

1.13.6 Depth first search on a directed graph with no loop

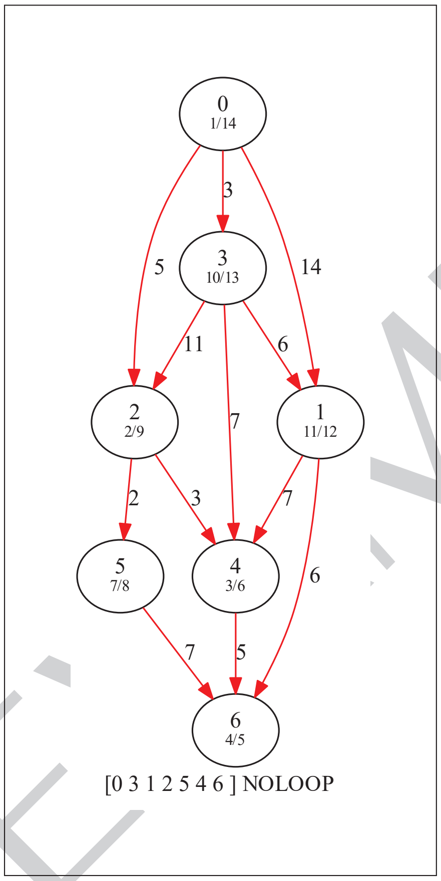


Figure 1.34: directed graph with no loop

```
## Jagadeesh Vasudevamurthy ####
## dot -Tpdf C:\scratch\outputs\dot\7dfs.dot -o C:\scratch\outputs\dot\7dfs.dot.pdf
digraph g {
   label = "[0 3 1 2 5 4 6 ] NOLOOP"
   0[label = <0<BR /><FONT POINT-SIZE="10">1/14</FONT>>]
   2[label = <2<BR /><FONT POINT-SIZE="10">2/9</FONT>>]
   3[label = <3<BR /><FONT POINT-SIZE="10">10/13</FONT>>]
   1[label = <1<BR /><FONT POINT-SIZE="10">11/12</FONT>>]
   6[label = <6<BR /><FONT POINT-SIZE="10">4/5</FONT>>]
   4[label = <4<BR /><FONT POINT-SIZE="10">3/6</FONT>>]
   5[label = <5<BR /><FONT POINT-SIZE="10">7/8</FONT>>]
edge [color=red]
   0 -> 2 [label = 5]
   0 -> 3 [label = 3]
   0 -> 1 [label = 14]
   2 -> 4 [label = 3]
   2 -> 5 [label = 2]
   3 -> 2 [label = 11]
   3 -> 1 [label = 6]
   3 -> 4 [label = 7]
   1 -> 6 [label = 6]
   1 -> 4 [label = 7]
   4 -> 6 [label = 5]
   5 -> 6 [label = 7]
}
```