18.5 Graph representation using fanins and fanouts lists

18.5.1 GraphInputOutputDirr

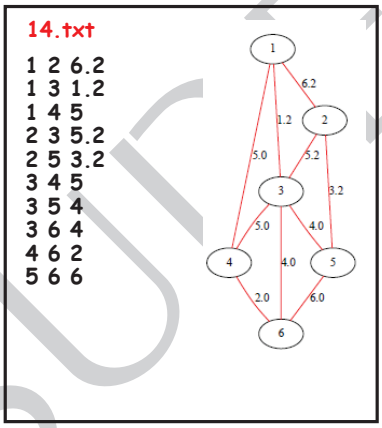
```
class GraphInputOutputDir {
  public static final String inputFileBase = "C:\\Users\\jag\\OneDrive\\vasu\\work\\algdata\\graphdata\\";
  public static final String outputFileBase = "C:\\scratch\\outputs\\dot\\";
  private static final String dot2pdfExec = "C:/Program Files (x86)/Graphviz2.38/bin/dot.exe";
  //make it null if you don't want to use dot2pdfExec
  //private static final String dot2pdfExec = "" ;


   static public void dot2pdf(String s) {}
```

```
C:\Users\jag\OneDrive\vasu\work\algdata\graphdata>ls
1.txt          12.txt  17.txt   2.txt   24.txt  6.txt   cat.txt    largeEWD.txt      scc1.txt      tinyEWG.txt  uw2.txt
10.txt         13.txt  18.txt   20.txt  3.txt   7.txt   hd1.txt    loopparallel.txt  td1.txt       tinyG.txt
10000EWD.txt   14.txt  18a.txt  21.txt  4.txt   8.txt   hd2.txt    mediumDG.txt      tinyDG.txt    u1.txt
1000EWD.txt    15.txt  19.txt   22.txt  5.txt   9.txt   hd3.txt    mediumEWD.txt     tinyEWD.txt   udf1.txt
11.txt         16.txt  19a.txt  23.txt  50.txt  README  krus1.txt  mediumG.txt       tinyEWD1.txt  uw1.txt


C:\scratch\outputs\dot>ls
13.dot  13.pdf  14.dot  14.pdf  15.dot  15.pdf  16.dot  16.pdf  cat.dot  cat.pdf  hd2.dot  loopparallel.dot  loopparallel.pdf

C:\scratch\outputs\dot>
```



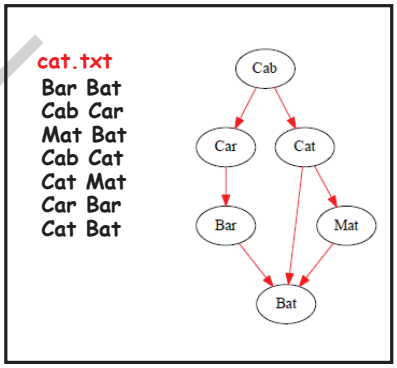


Figure 18.14: GraphInputOutputDir class

18.5.2 class GraphType


```
class GraphType {
  public enum Type {
    NONE, UNDIRECTED , DIRECTED, WEIGHTED_UNDIRECTED, WEIGHTED_DIRECTED
  }

  static String gtype(GraphType.Type t) {
    if (t == GraphType.Type.UNDIRECTED) {
      return "UNDIRECTED";
    }
    if (t == GraphType.Type.DIRECTED) {
      return "DIRECTED";
    }
    if (t == GraphType.Type.WEIGHTED_UNDIRECTED) {
      return "WEIGHTED_UNDIRECTED";
    }
    if (t == GraphType.Type.WEIGHTED_DIRECTED) {
      return "WEIGHTED_DIRECTED";
    }
    return "NONE";
  }
}
```

Figure 18.15: GraphType class

18.5.3 class GraphIO

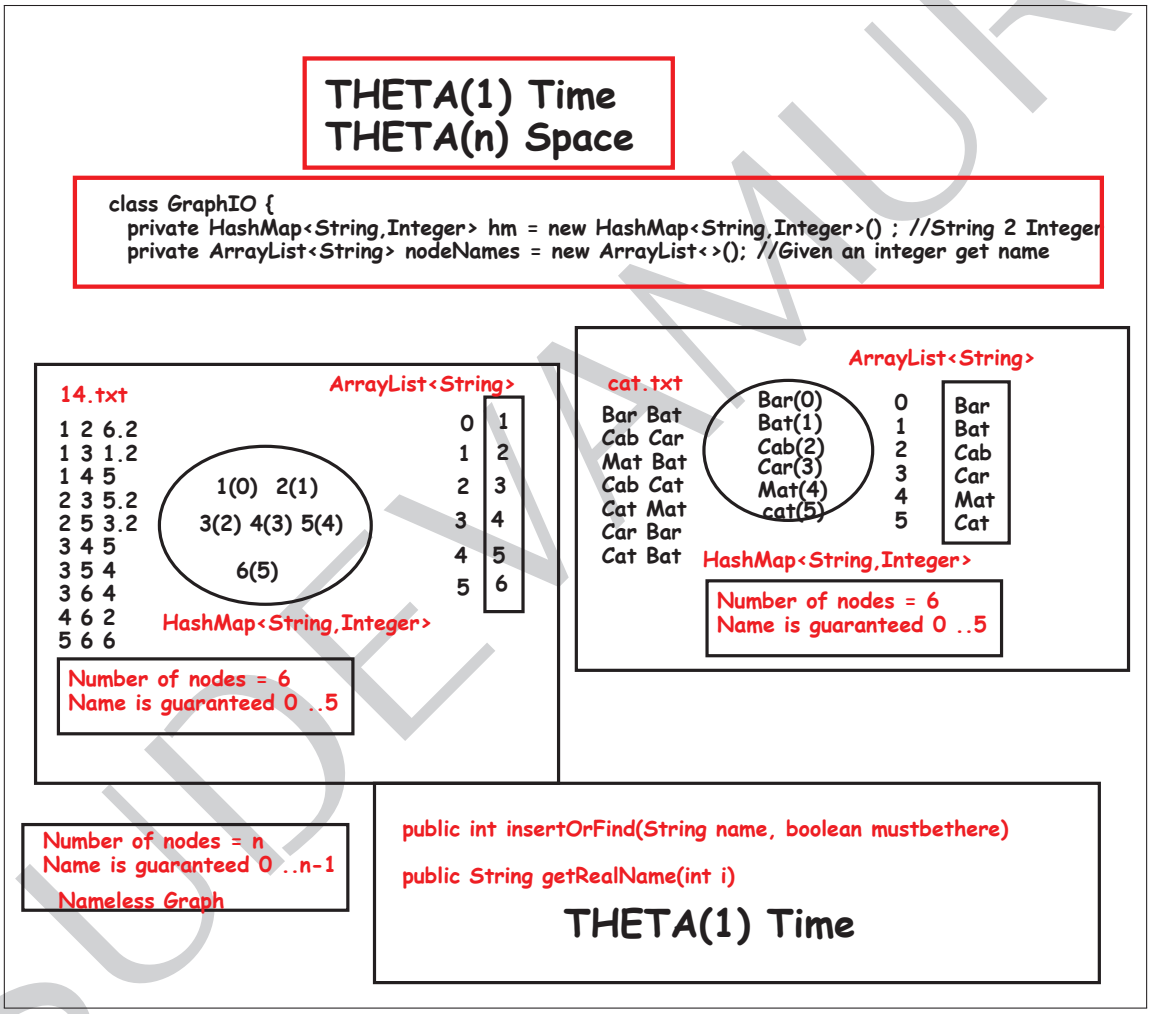


Figure 18.16: GraphIO class

18.5.4 class Graph

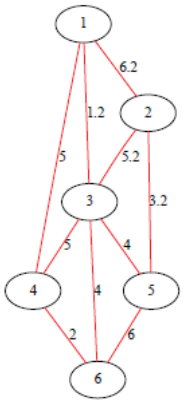

```
class Edge {
  public int other ;
  public double cost ;
  Edge(int other, double cost) {
    this.other = other ;
    this.cost = cost ;
  }
}

class Node {
  public int num ;
  public HashMap<Integer, Edge> fanout ; // Key is int. Value is Edge
  public HashMap<Integer, Edge> fanin ; // Key is int. Value is Edge

  Node(int n) {
    this.num = n ;
    fanout = new HashMap<Integer,Edge>();
    fanin =  new HashMap<Integer,Edge>();
  }
}

class Graph{
  public GraphType.Type type; //Type of the graph
  public GraphIO io; //input/output
  public ArrayList<Node> nodes ; //Array of all nodes
  public int numEdges ;
  public IntUtil u = new IntUtil();

  Graph(GraphType.Type type, GraphIO io) {
    this.type = type ;
    this.io = io ;
    nodes = new ArrayList<Node>() ;
    numEdges = 0 ;
  }
}
```




```
GraphType.WEIGHTED_UNDIRECTED
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
```

Time: O(V + 2E)
Space:O(V + 2E)

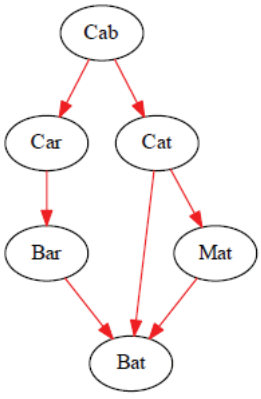
Figure 18.17: Representation of undirected weighted graph


```
class Edge {
  public int other ;
  public double cost ;
  Edge(int other, double cost) {
    this.other = other ;
    this.cost = cost ;
  }
}
class Node {
  public int num ;
  public HashMap<Integer, Edge> fanout ; // Key is int. Value is Edge
  public HashMap<Integer, Edge> fanin ; // Key is int. Value is Edge

  Node(int n) {
    this.num = n ;
    fanout = new HashMap<Integer,Edge>();
    fanin =  new HashMap<Integer,Edge>();
  }
}

class Graph{
  public GraphType.Type type; //Type of the graph
  public GraphIO io; //input/output
  public ArrayList<Node> nodes ; //Array of all nodes
  public int numEdges ;
  public IntUtil u = new IntUtil();

  Graph(GraphType.Type type, GraphIO io) {
    this.type = type ;
    this.io = io ;
    nodes = new ArrayList<Node>() ;
    numEdges = 0 ;
  }
}
```




```
GraphType.DIRECTED
Num Vertices = 6
Num Edges    = 7
Bar Fanouts: Bat
Bar FanIns: Car
Bat Fanouts: NONE
Bat FanIns: Bar,Mat,Cat
Cab Fanouts: Car,Cat
Cab FanIns: NONE
Car Fanouts: Bar
Car FanIns: Cab
Mat Fanouts: Bat
Mat FanIns: Cat
Cat Fanouts: Mat,Bat
Cat FanIns: Cab
```

TIME:O(V + E)
Space:O(V + E)

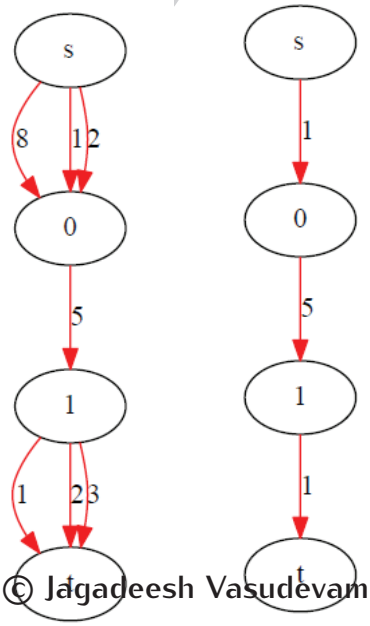
Figure 18.18: Representation of directed graph


```
class Edge {
  public int other ;
  public double cost ;
  Edge(int other, double cost) {
    this.other = other ;
    this.cost = cost ;
  }
}
class Node {
  public int num ;
  public HashMap<Integer, Edge> fanout ; // Key is int. Value is Edge
  public HashMap<Integer, Edge> fanin ; // Key is int. Value is Edge

  Node(int n) {
    this.num = n ;
    fanout = new HashMap<Integer,Edge>();
    fanin =  new HashMap<Integer,Edge>();
  }
}

class Graph{
  public GraphType.Type type; //Type of the graph
  public GraphIO io; //input/output
  public ArrayList<Node> nodes ; //Array of all nodes
  public int numEdges ;
  public IntUtil u = new IntUtil();

  Graph(GraphType.Type type, GraphIO io) {
    this.type = type ;
    this.io = io ;
    nodes = new ArrayList<Node>() ;
    numEdges = 0 ;
  }
}
```




```
GraphType.WEIGHTED_DIRECTED
Num Vertices = 4
Num Edges    = 3
s Fanouts: 0
s FanIns: NONE
0 Fanouts: 1
0 FanIns: s
1 Fanouts: t
1 FanIns: 0
t Fanouts: NONE
t FanIns: 1
```

Figure 18.19: Representation of directed weighted graph that has parallel edges

18.5.5 How to use *class Graph*

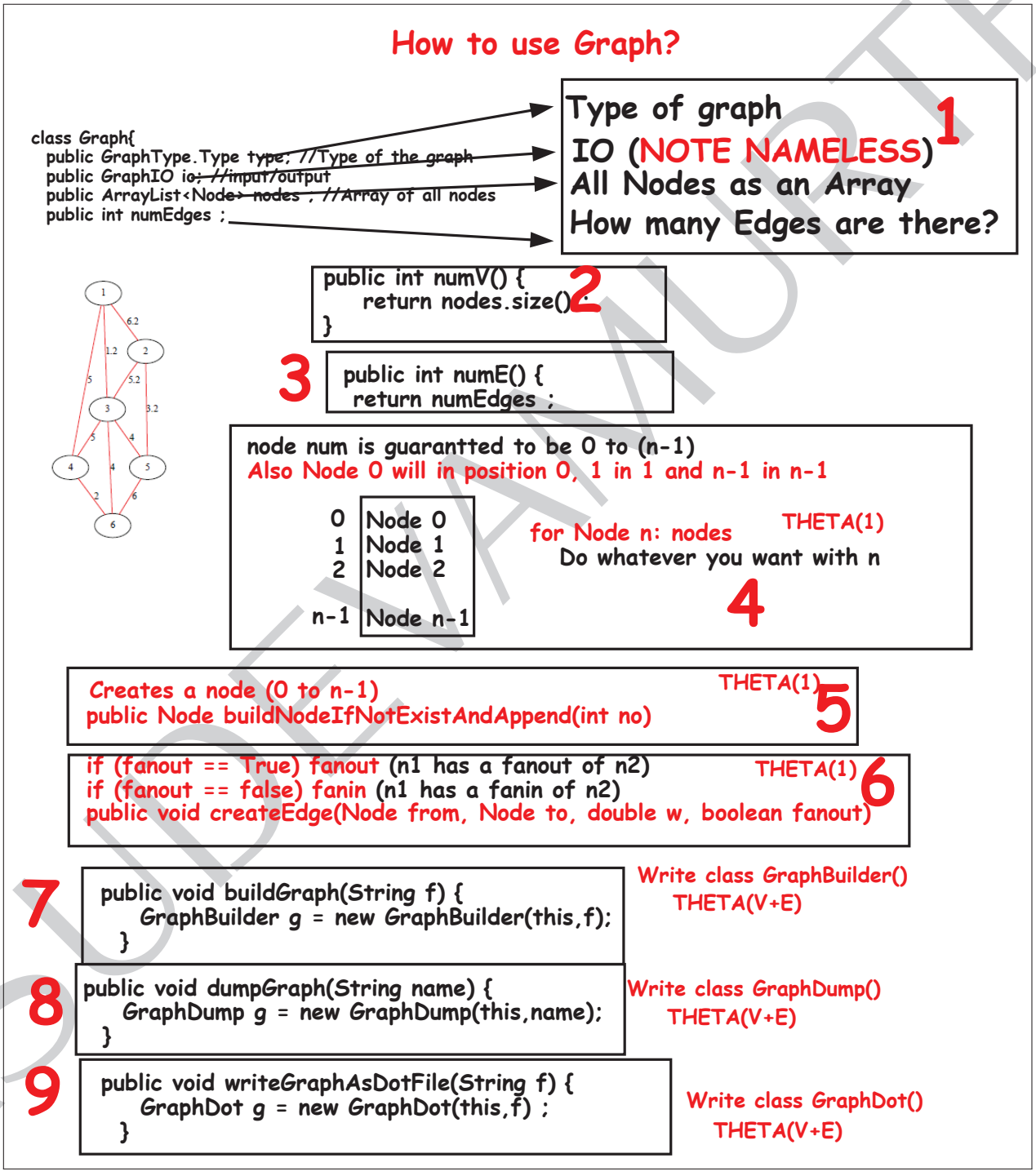


Figure 18.20: Public functions of Graph

18.5.6 How to get *fanouts* and *fanins* edges from a Node *node*

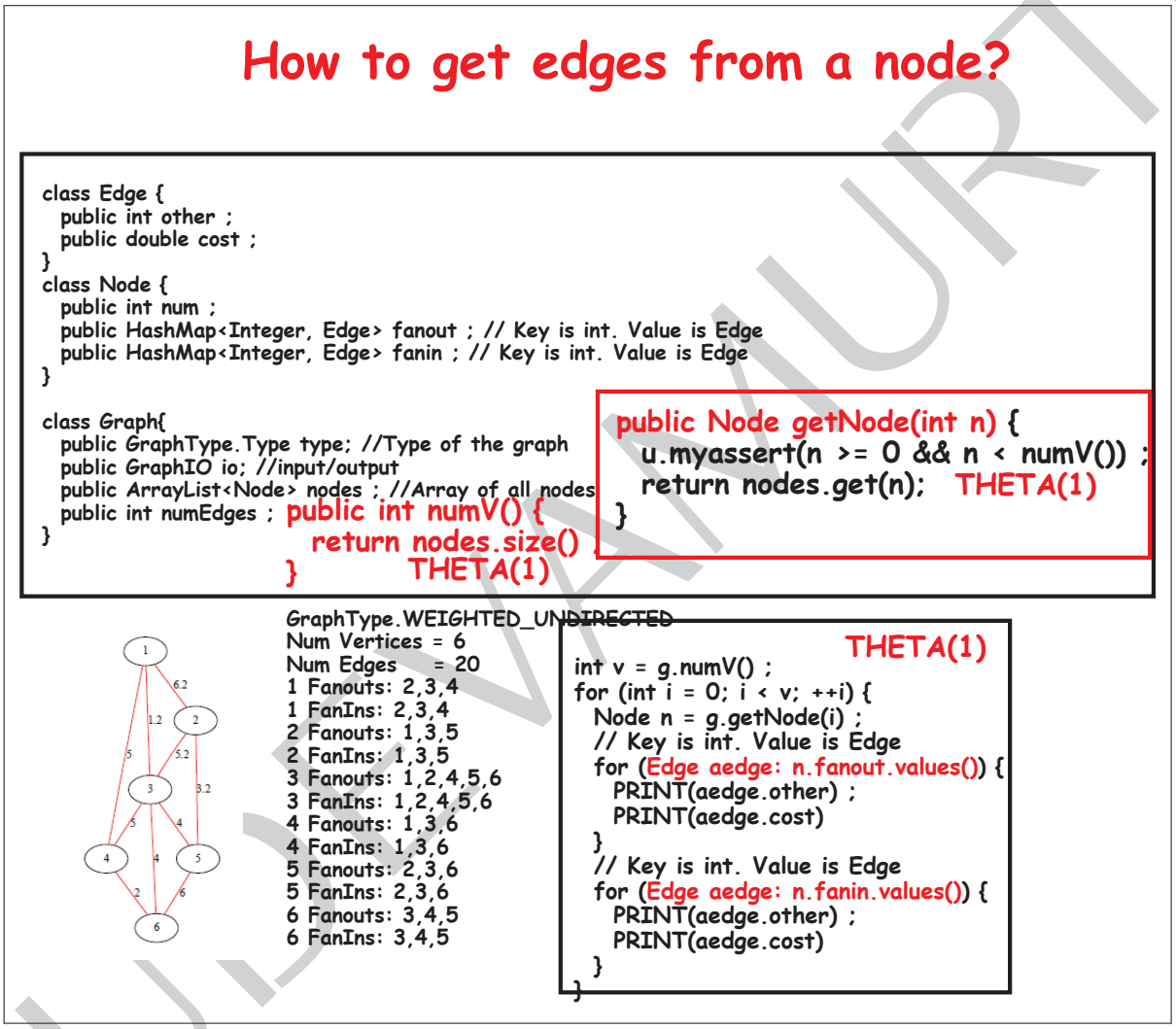


Figure 18.21: How to get *fanouts* and *fanins* edges from a Node *node*

18.5.7 How to know a node has an edge *e(int)*

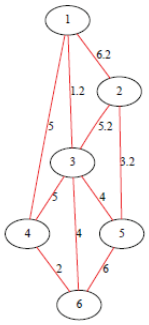
How to know a node has an edge e(int) ?

```
class Edge {
  public int other ;
  public double cost ;
```

```
class Node {
  public int num ;
  public HashMap<Integer, Edge> fanout ; // Key is int. Value is Edge
  public HashMap<Integer, Edge> fanin ; // Key is int. Value is Edge

   // Does this node has a fanout of 'n'
  // Time: THETA(1) Space; THETA(1)
  Edge hasAFanoutEdge(int n) {
    // Key is int. Value is Edge
    Edge stored = fanout.get(n); //O(1)
    return stored ; //NULL if not there
  }

  //Does this node has a fanin of 'n'
  // Time: THETA(1) Space; THETA(1)
  Edge hasAFaninEdge(int n) {
    // Key is int. Value is Edge
    Edge stored = fanin.get(n); //O(1)
    return stored ; //NULL if not there
  }
```




```
GraphType.WEIGHTED_UNDIRECTED
Num Vertices = 6
Num Edges    = 20
n1 1 Fanouts: 2,3,4
n1 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
```

```
edge = n1.hasAFanoutEdge(3)
returns an edge
edge = n1.hasAFanoutEdge(5)
returns None
```

```
edge = n1.hasAFaninEdge(3)
returns an edge
edge = n1.hasAFaninEdge(5)
returns None
```

Figure 18.22: How to know a node has an edge *e(int)*

18.6 Build a graph from a file

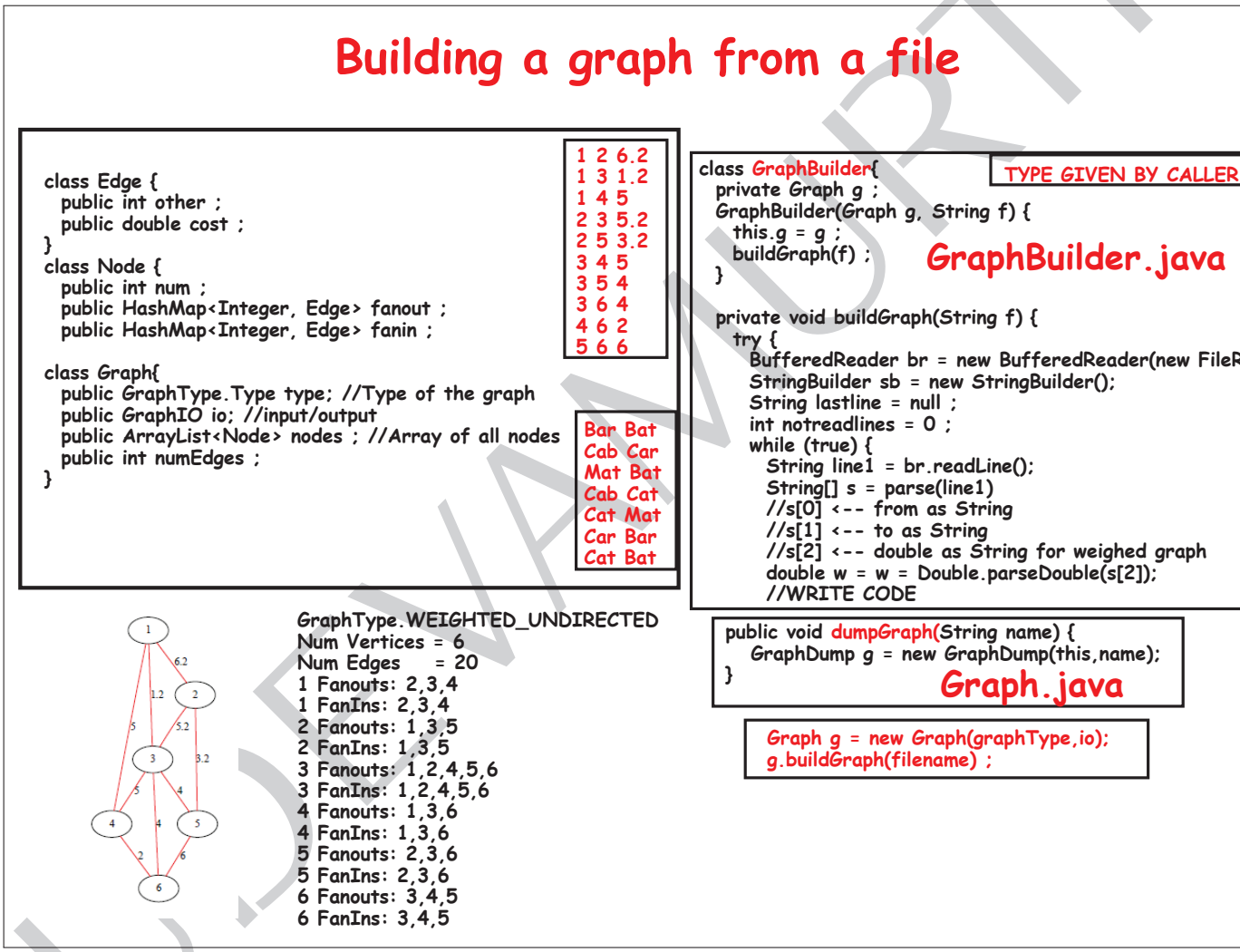


Figure 18.23: Build a graph from a file. Graph type is already known

18.7 Write a graph as a text file

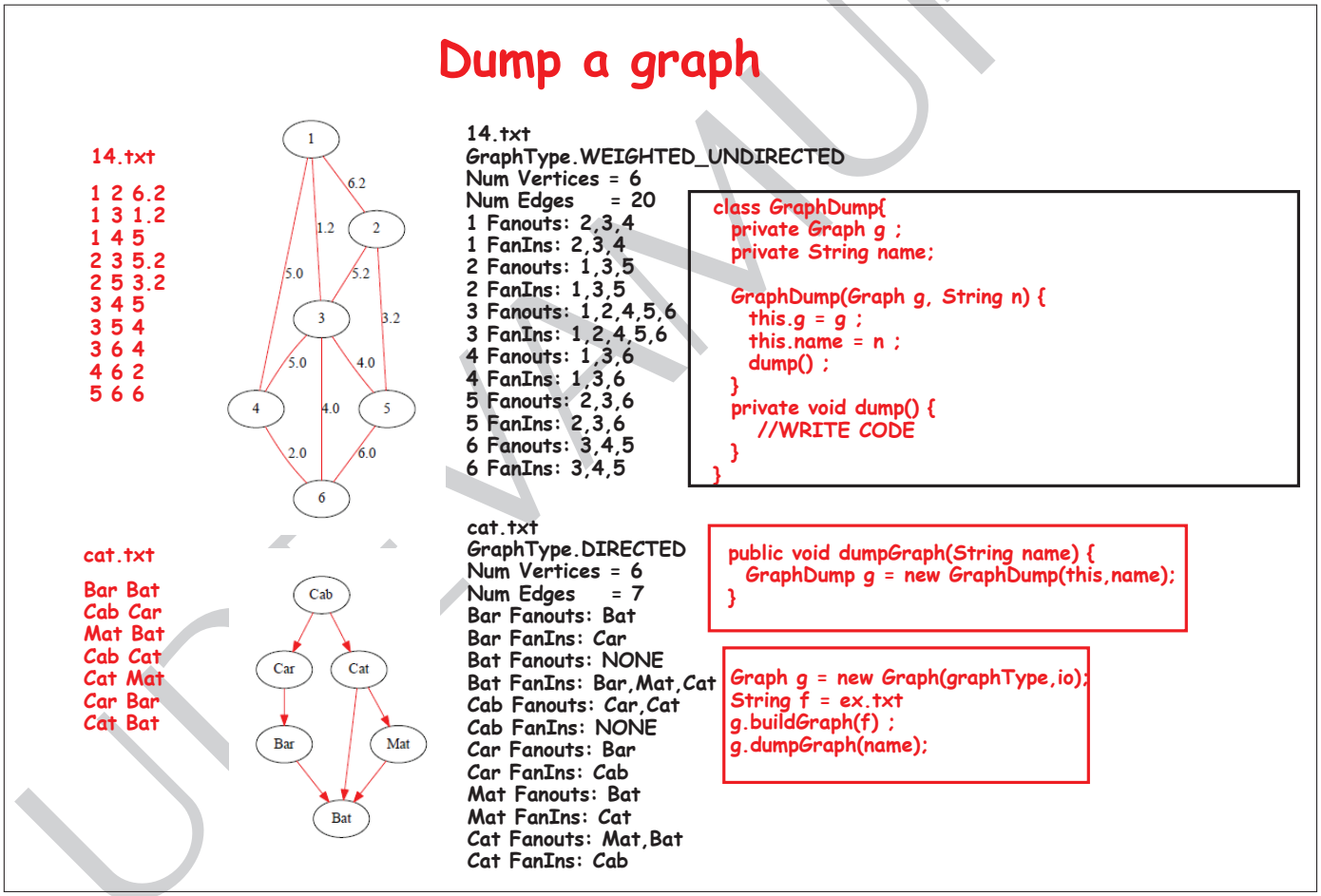


Figure 18.24: Write a graph as a text file

18.8 Write a graph as a dot file

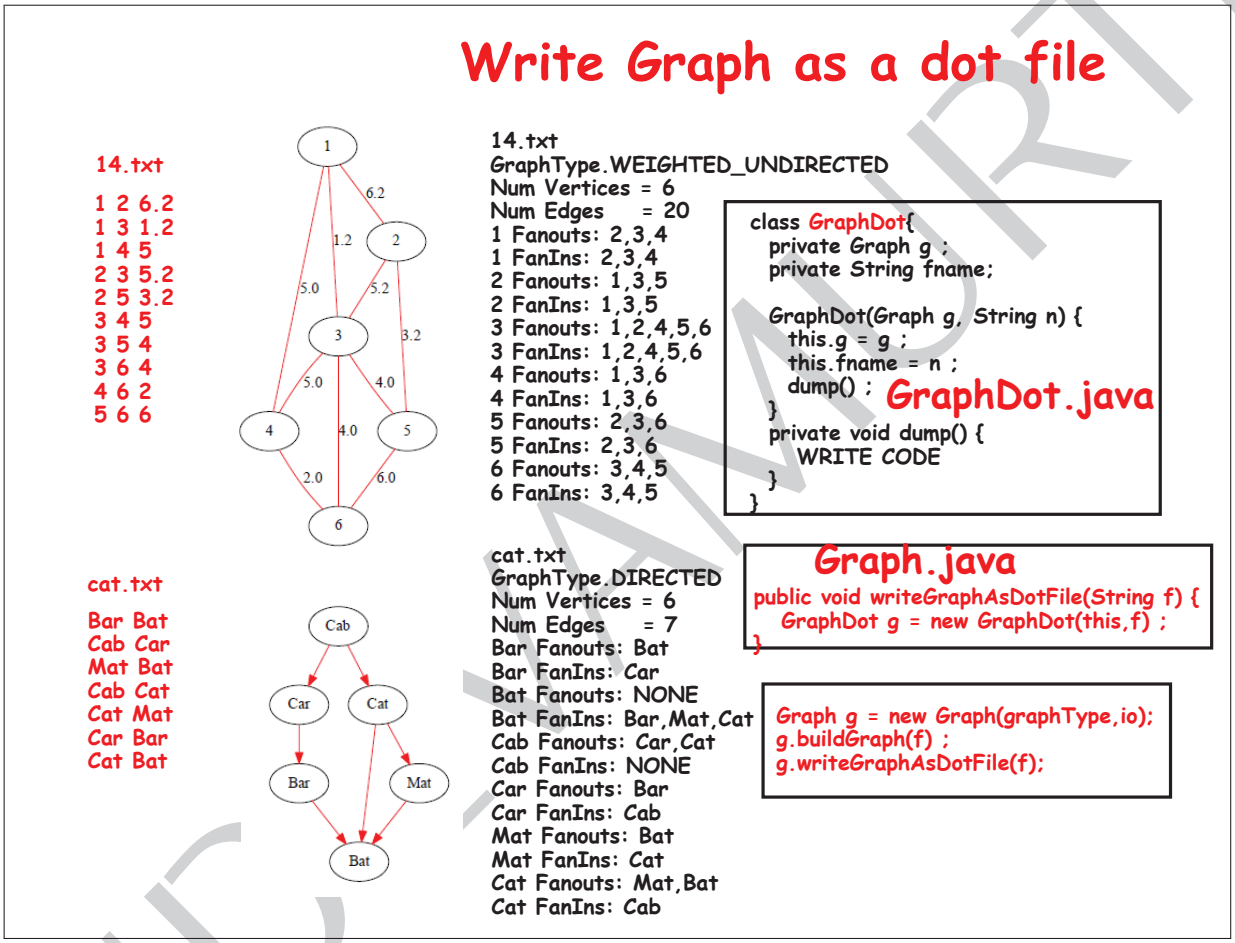


Figure 18.25: Write a graph as a dot file

18.8.1 How to generate *dot* files and *pdf* files

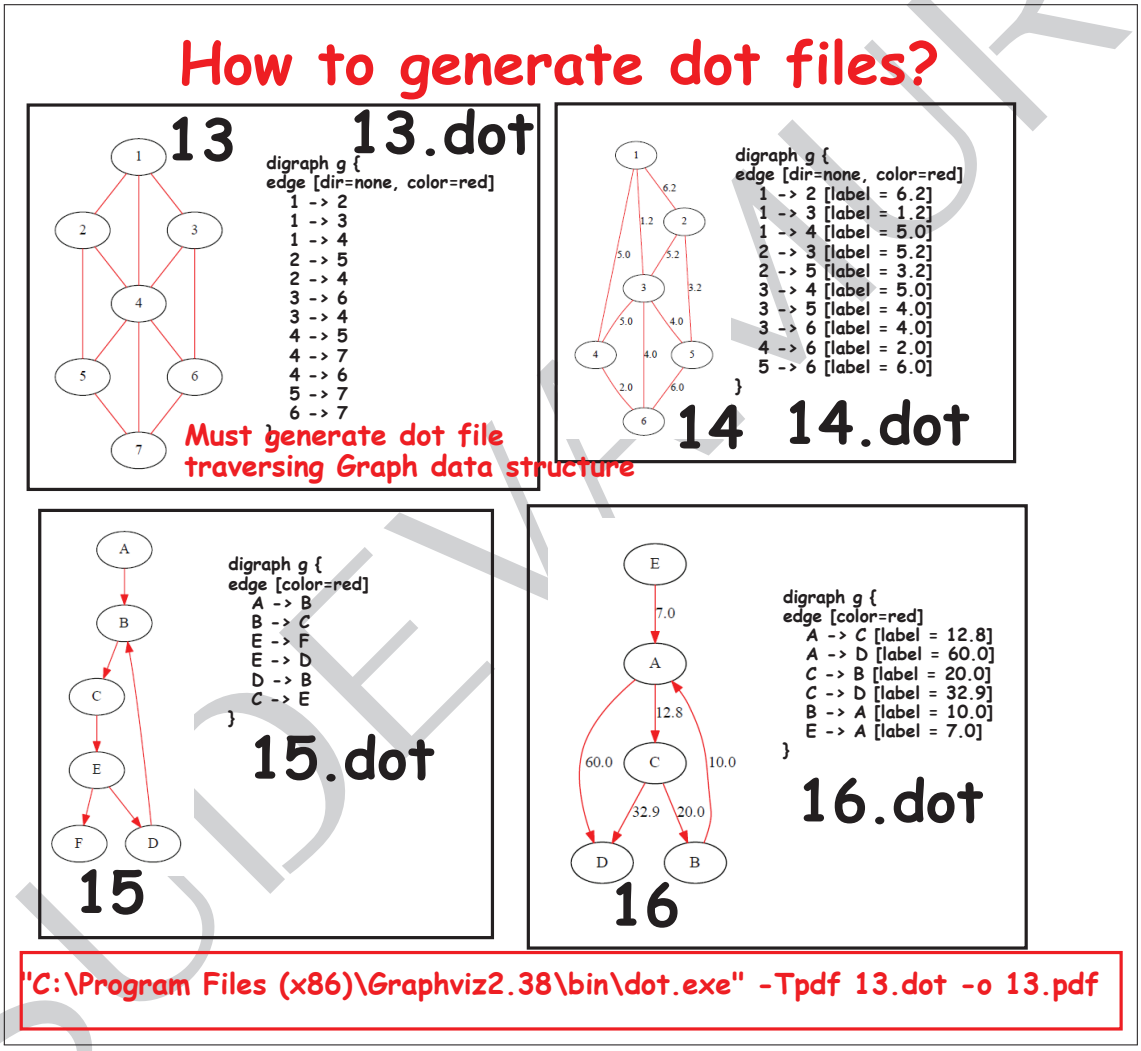


Figure 18.26: How to generate *dot* files and *pdf* files

