CSC 501/401

Lectures on

Analysis of Algorithms

by

Dr. Marek A. Suchenek ©

Computer Science
CSUDH

Copyrighted material

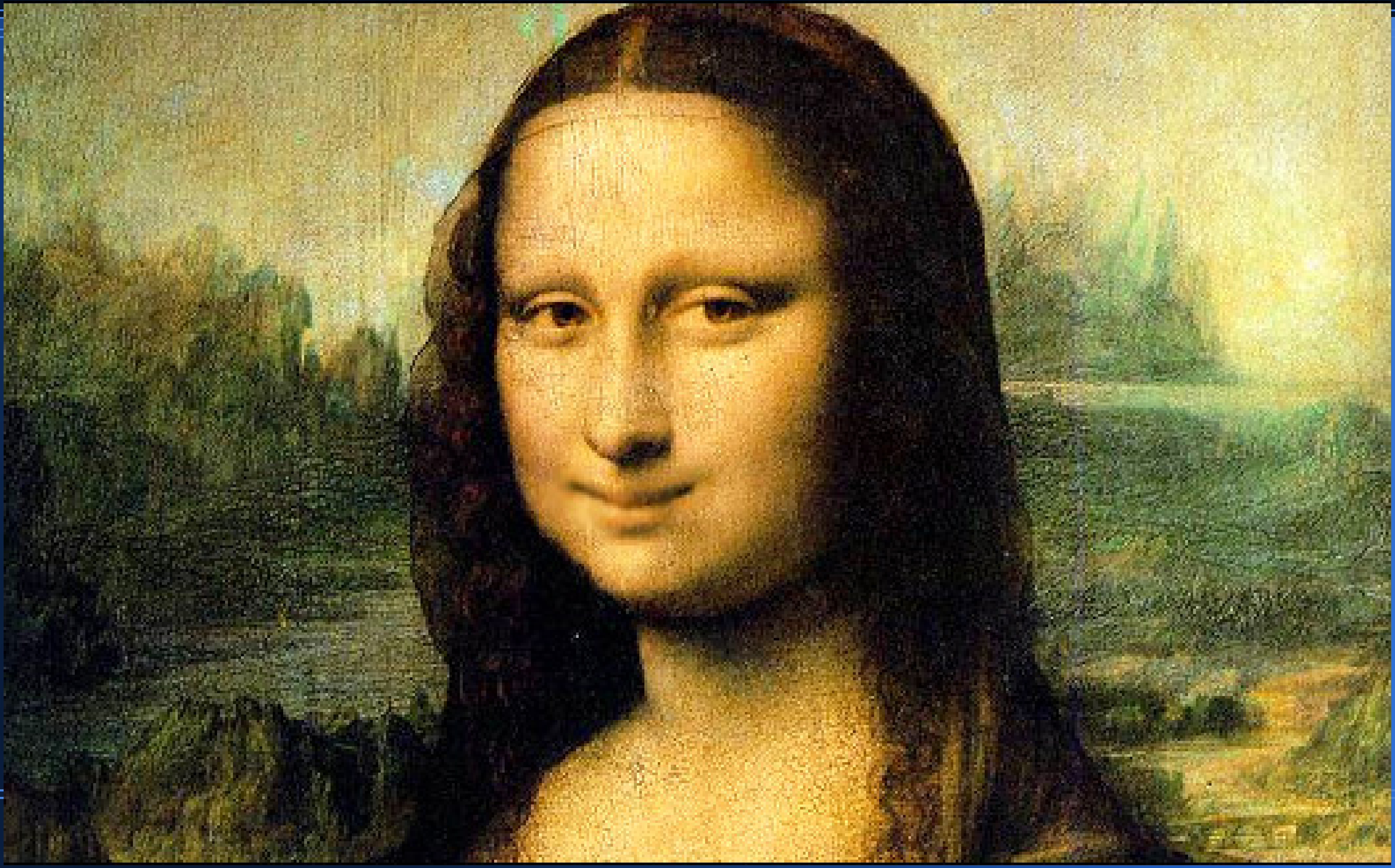
CSC 501/401

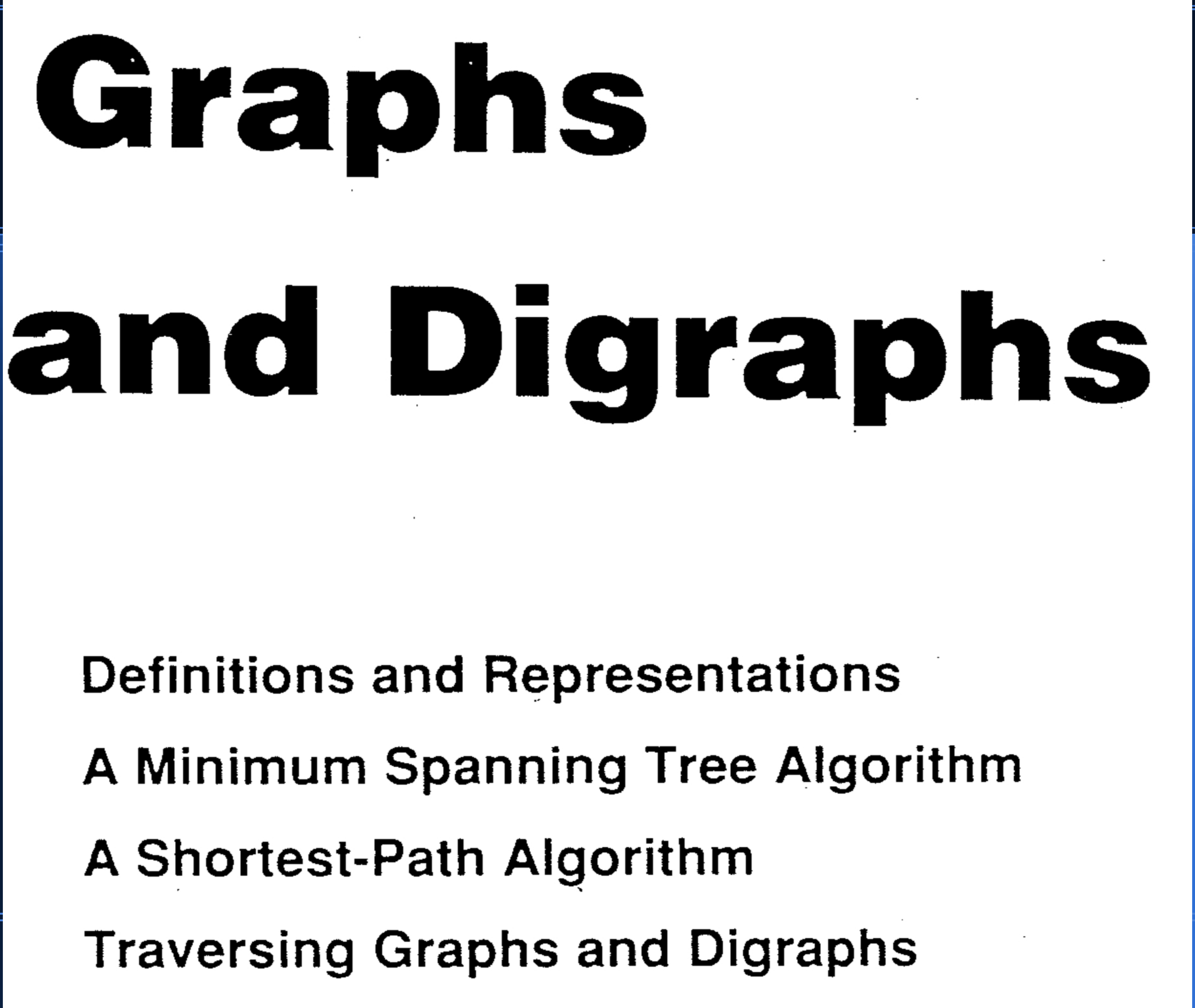
Chapter 7

Graphs and Graph Traversals

Depth-First Search
Breadth-First Search

This will not be covered by Test 2





Graphs and Digraphs

Definitions and Representations

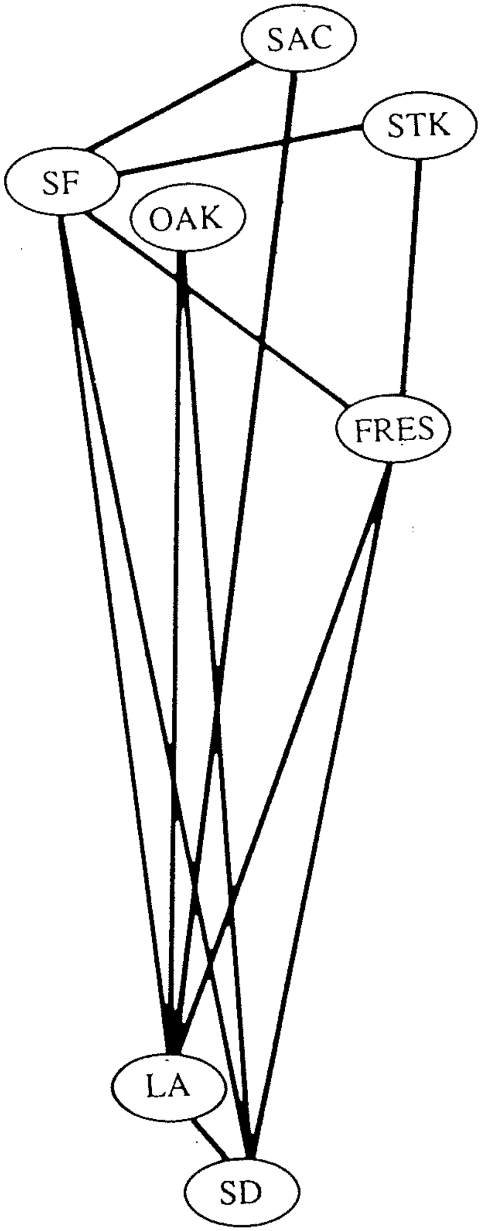
A Minimum Spanning Tree Algorithm

A Shortest-Path Algorithm

Traversing Graphs and Digraphs

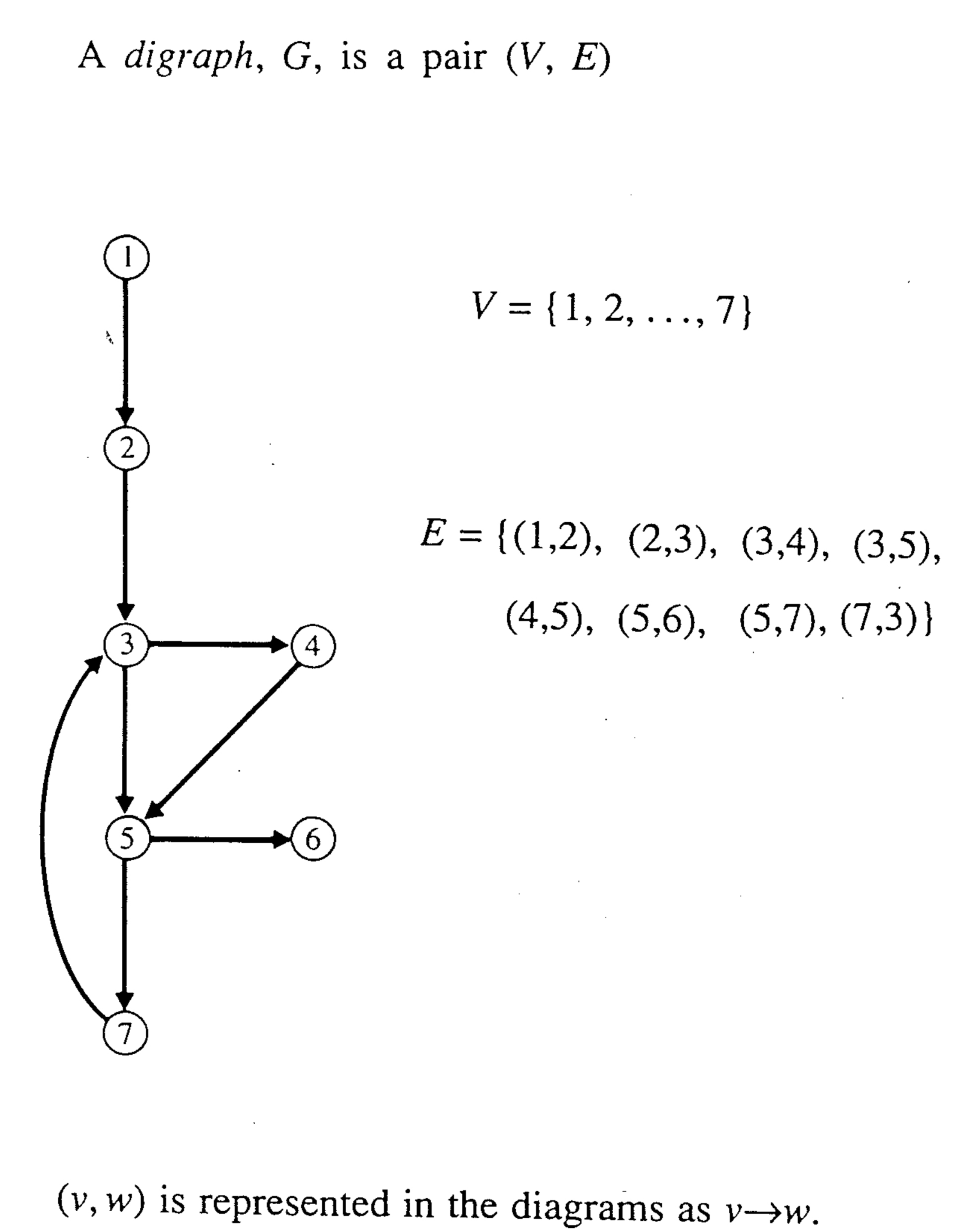
Definitions and Examples

graph, G, is a pair (V, E)



$V = \{\text{SF, OAK, SAC, STK, FRES, LA, SD}\}$

$E = \{\{\text{SF, STK}\}, \{\text{SF, SAC}\}, \{\text{SF, LA}\}, \{\text{SF, SD}\}, \{\text{SF, FRES}\}, \{\text{SD, OAK}\}, \{\text{SAC, LA}\}, \{\text{LA, OAK}\}, \{\text{LA, FRES}\}, \{\text{LA, SD}\}, \{\text{FRES, STK}\}, \{\text{SD, FRES}\}\}.$



A *digraph*, G, is a pair (V, E)

$$V = \{1, 2, \ldots, 7\}$$

$$E = \{(1,2),\ (2,3),\ (3,4),\ (3,5),\ (4,5),\ (5,6),\ (5,7),\ (7,3)\}$$

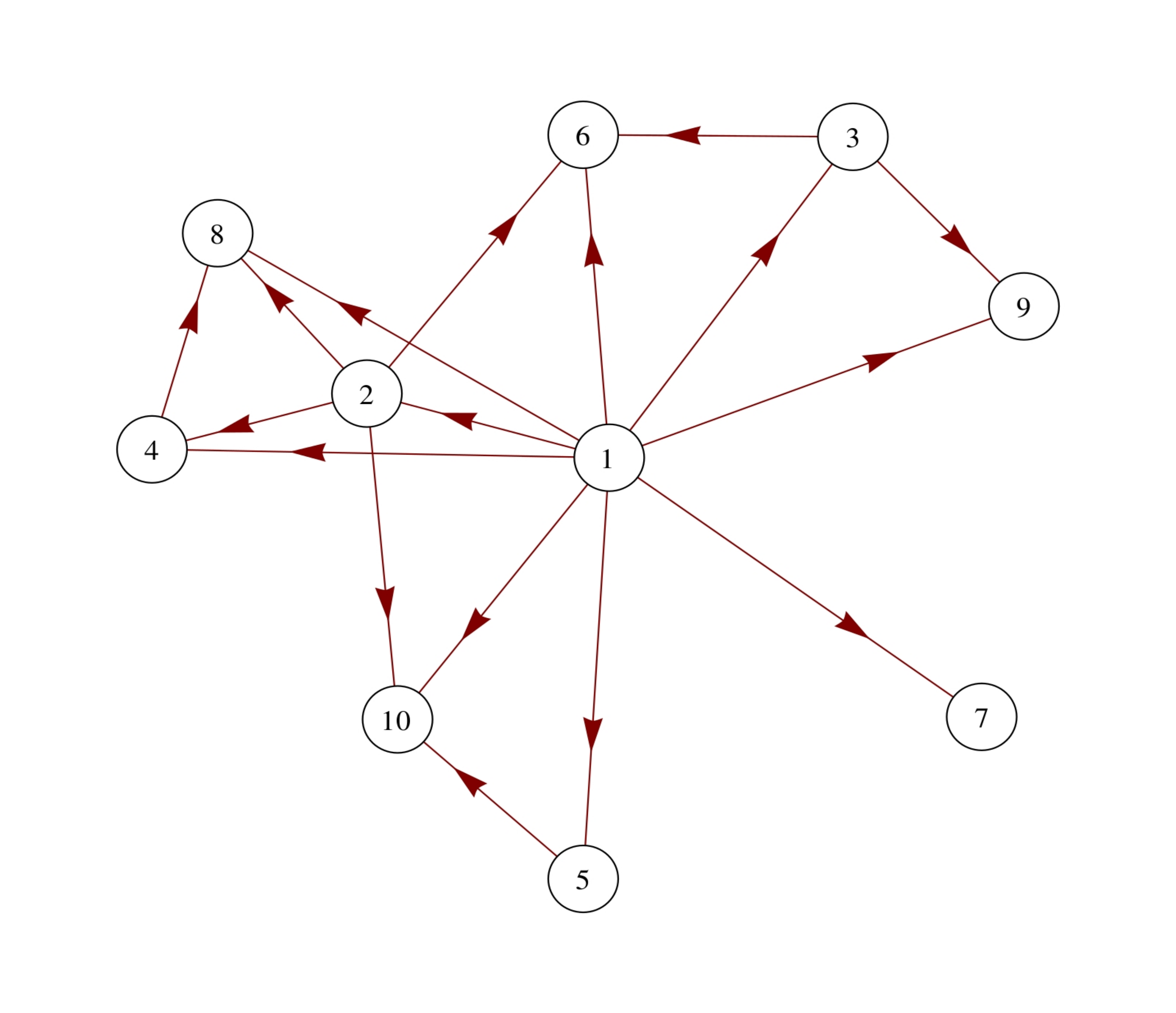
(v, w) is represented in the diagrams as $v \rightarrow w$.

(x,y)

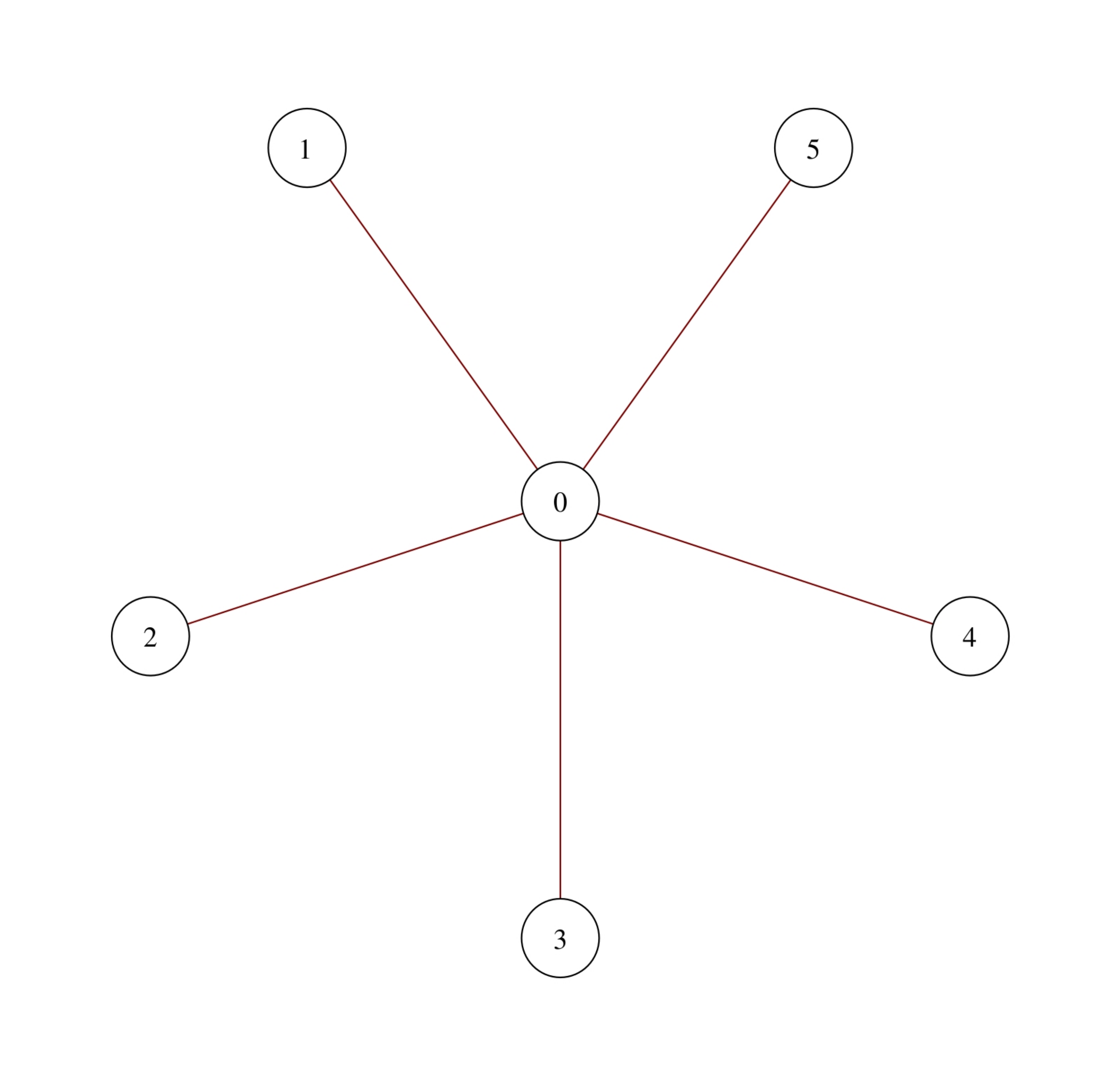
in E

iff

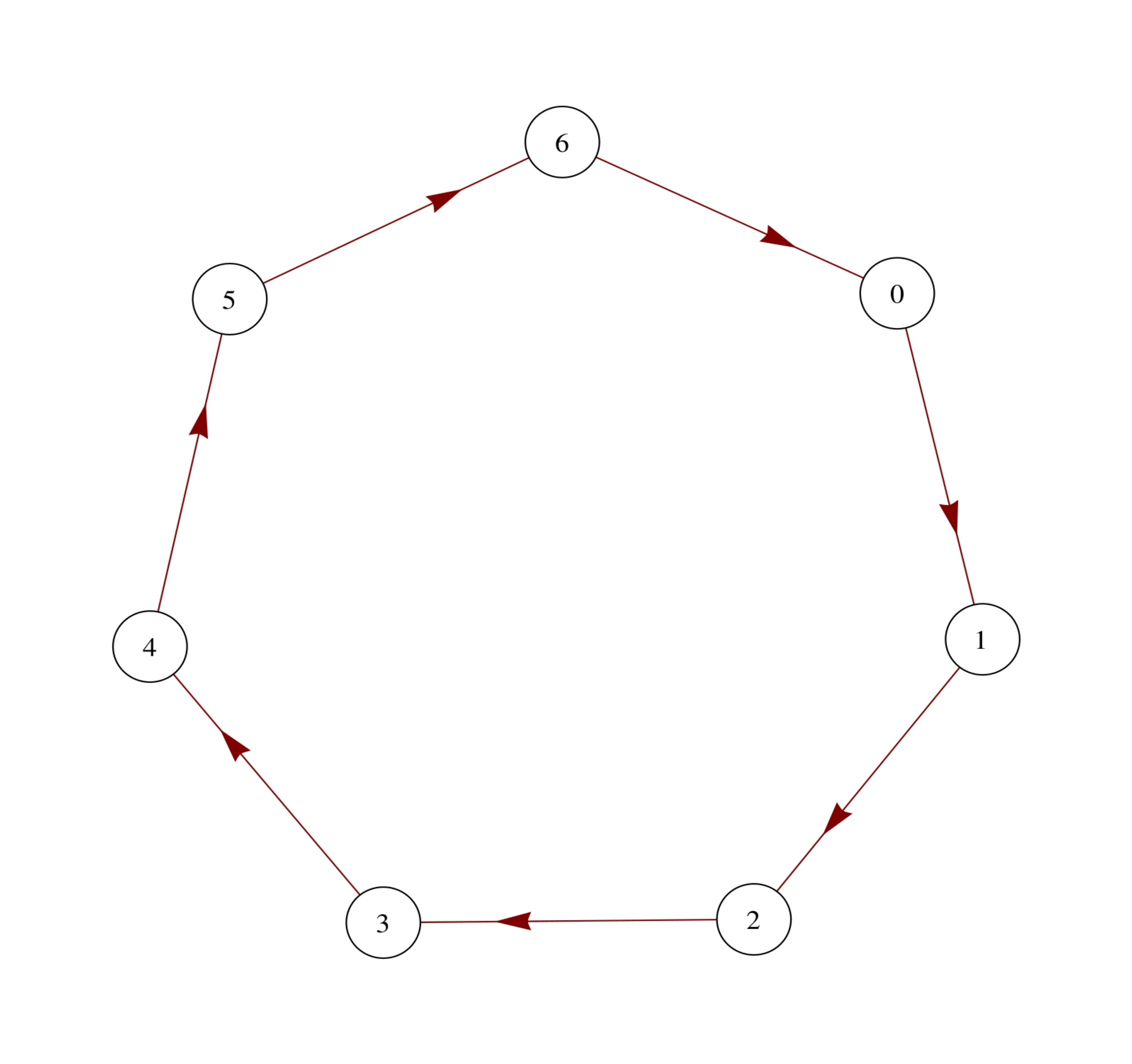
y|x



Star

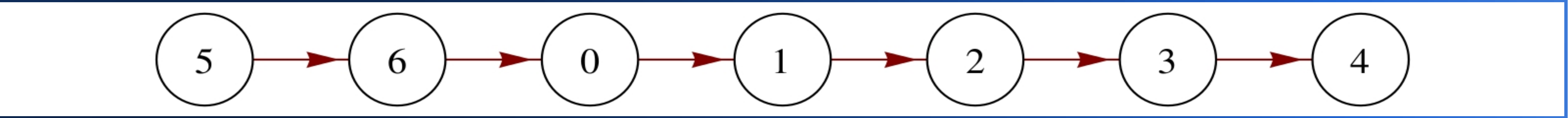


Cycle



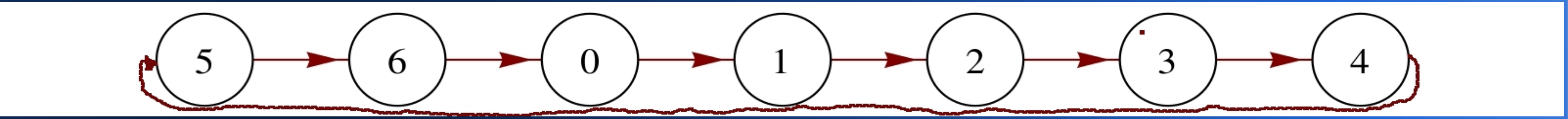
Graphs and Digraphs

Cycle



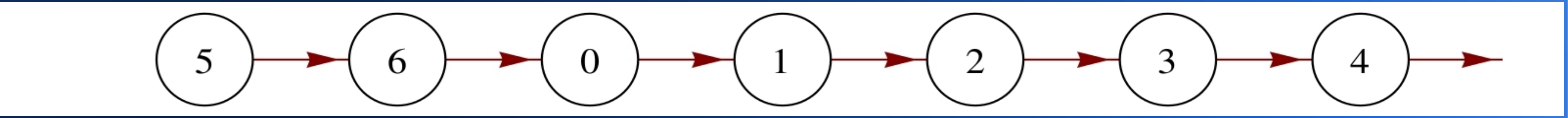
Graphs and Digraphs

Cycle



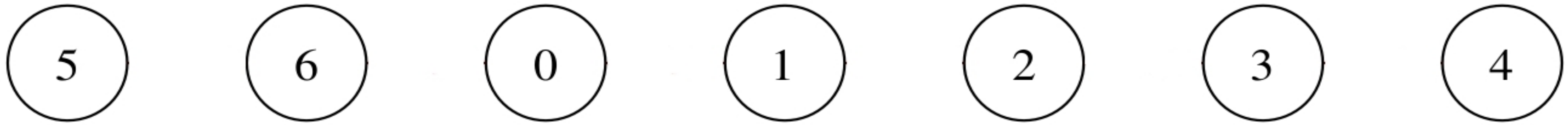
Graphs and Digraphs

Cycle



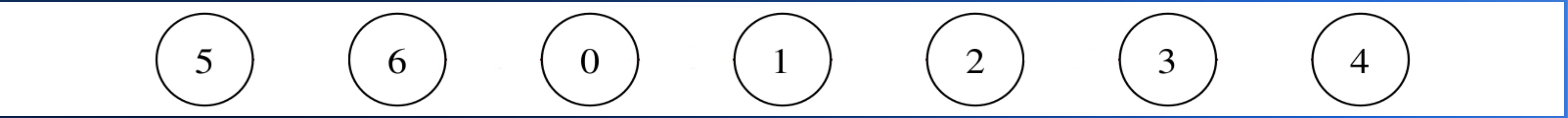
Graphs and Digraphs

Cycle



Graphs and Digraphs

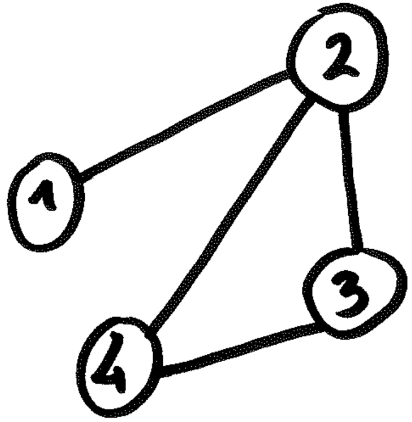
Cycle



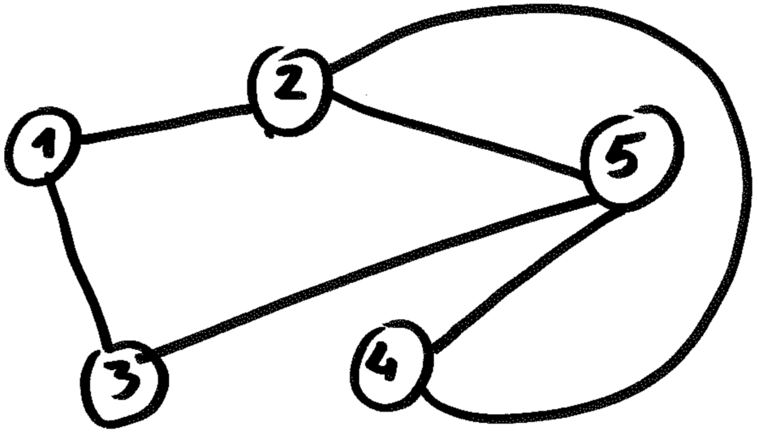
(5, 6, 0, 1, 2 ,3 ,4)

Gra ns

GRAPH



SUBGRAPH



PATH: 5 3 1 2 4

CYCLE: 5 3 1 2 4

Finding Connected Components of a Graph

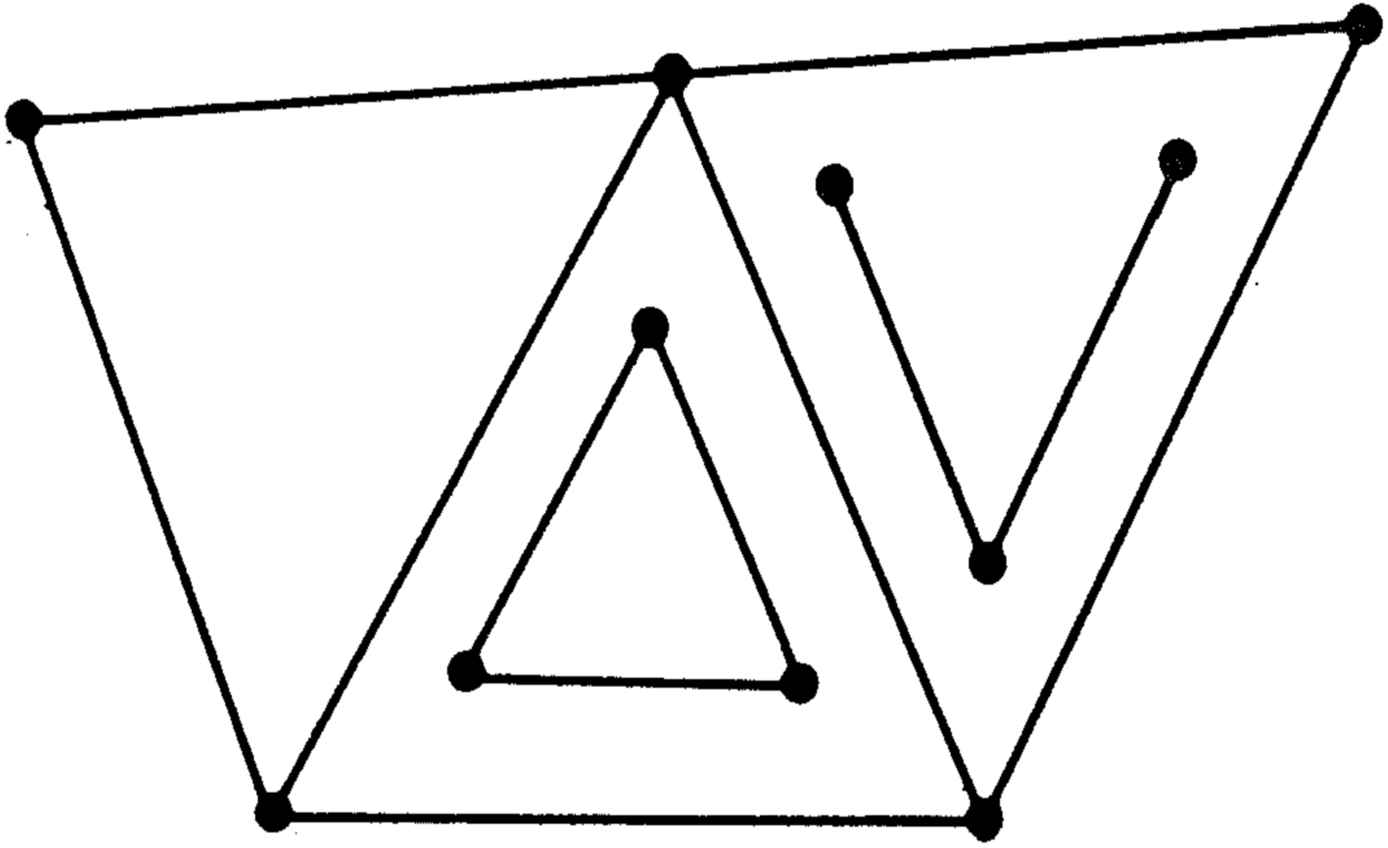
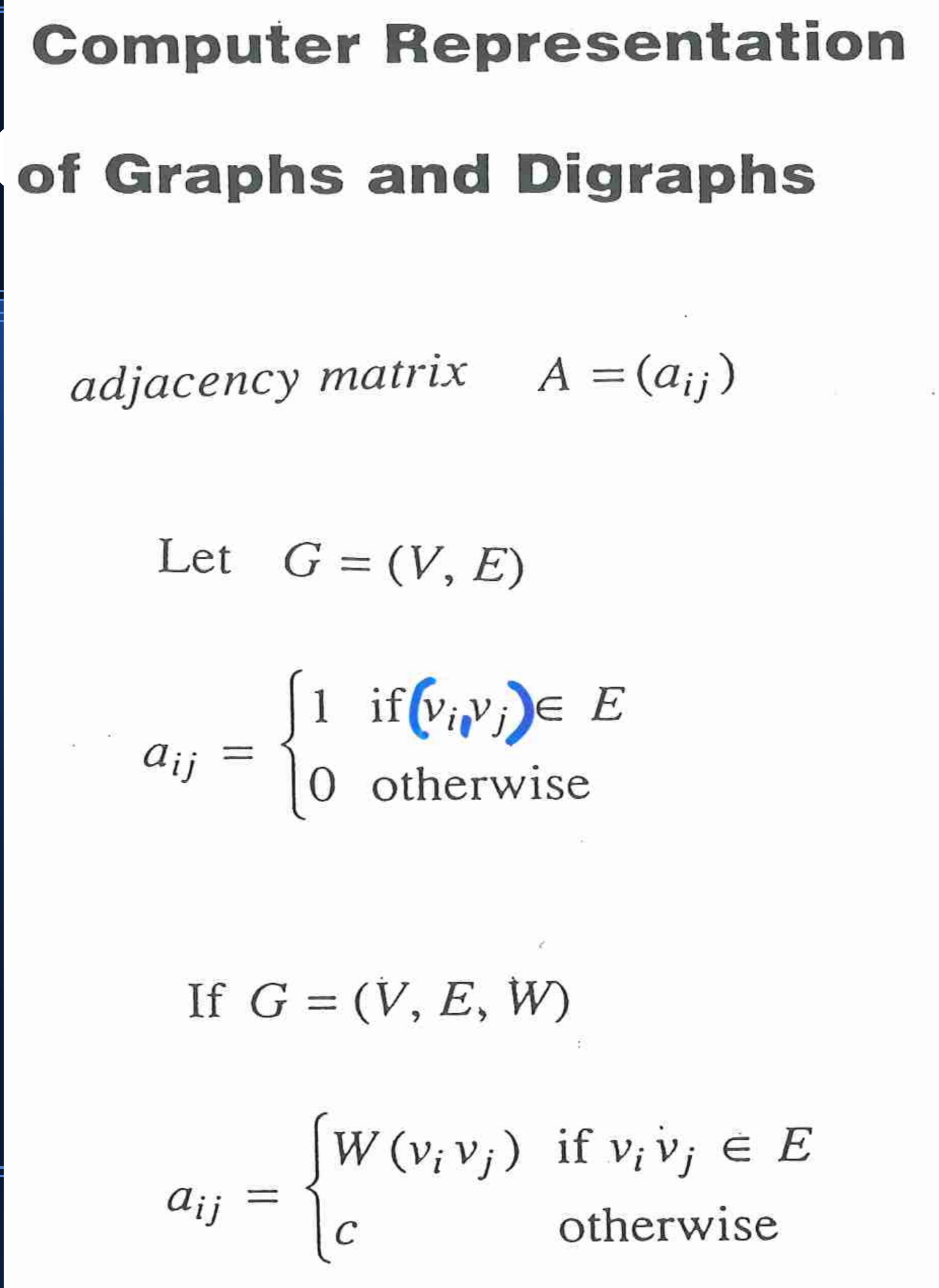


Figure 4.7 A graph with three connected components.



Computer Representation of Graphs and Digraphs

adjacency matrix $A = (a_{ij})$

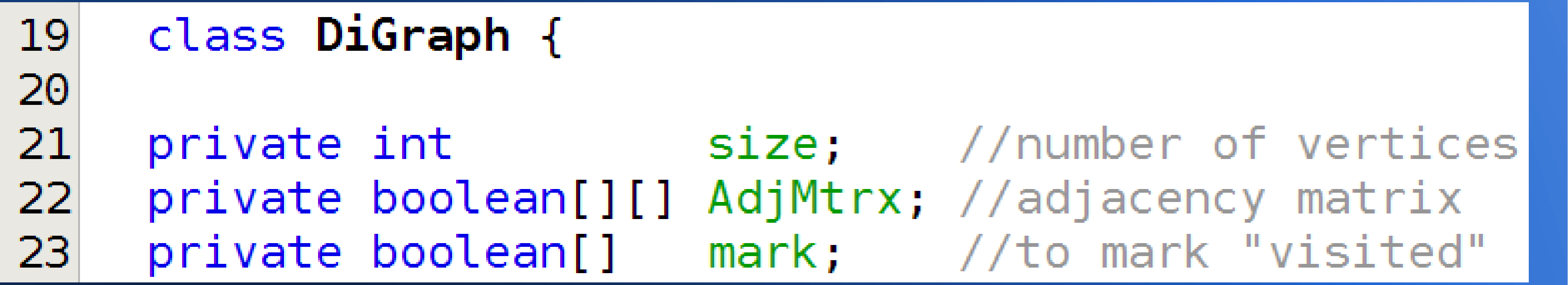
Let $G = (V, E)$

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

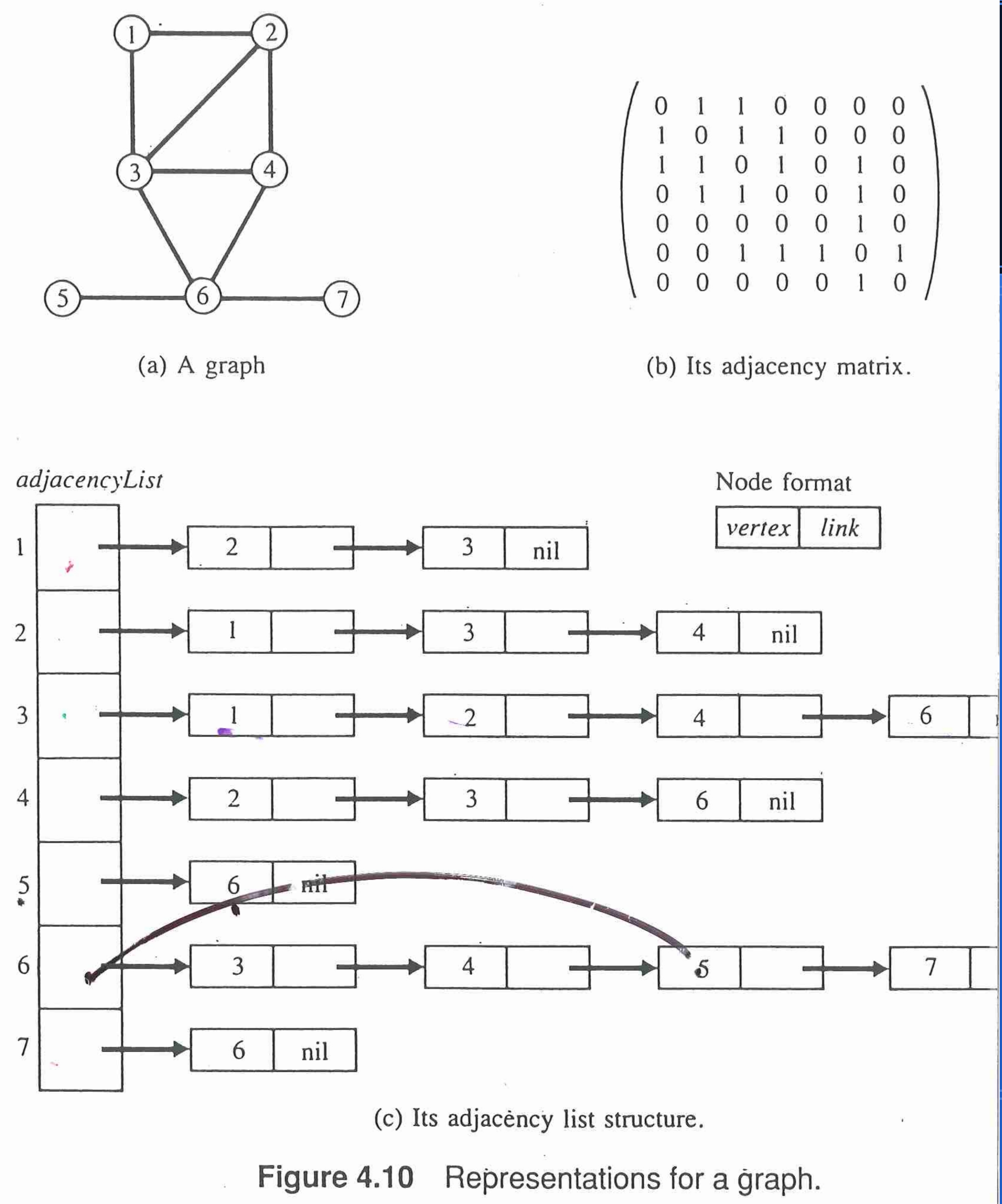
If $G = (V, E, W)$

$$a_{ij} = \begin{cases} W(v_i v_j) & \text{if } v_i v_j \in E \\ c & \text{otherwise} \end{cases}$$

Graphs and Digraphs



```
class DiGraph {

private int         size;    //number of vertices
private boolean[][] AdjMtrx; //adjacency matrix
private boolean[]   mark;    //to mark "visited"
```



(a) A graph

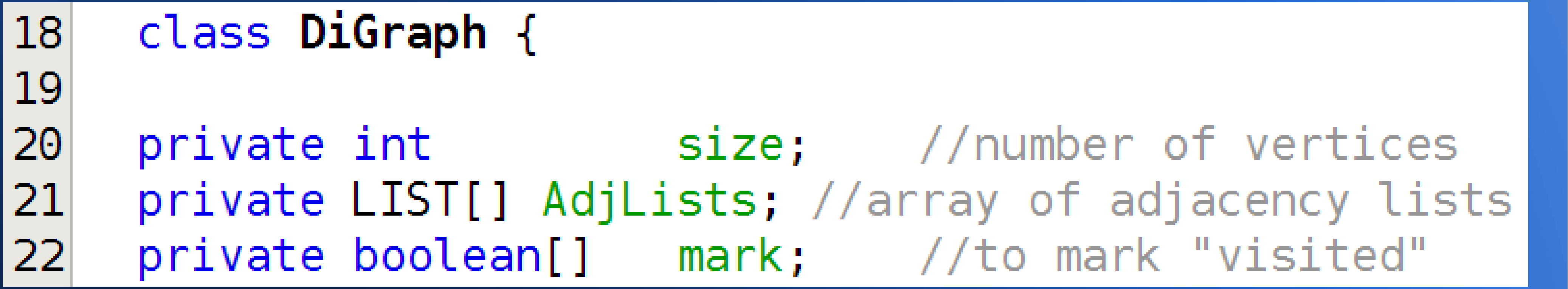
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) Its adjacency matrix.

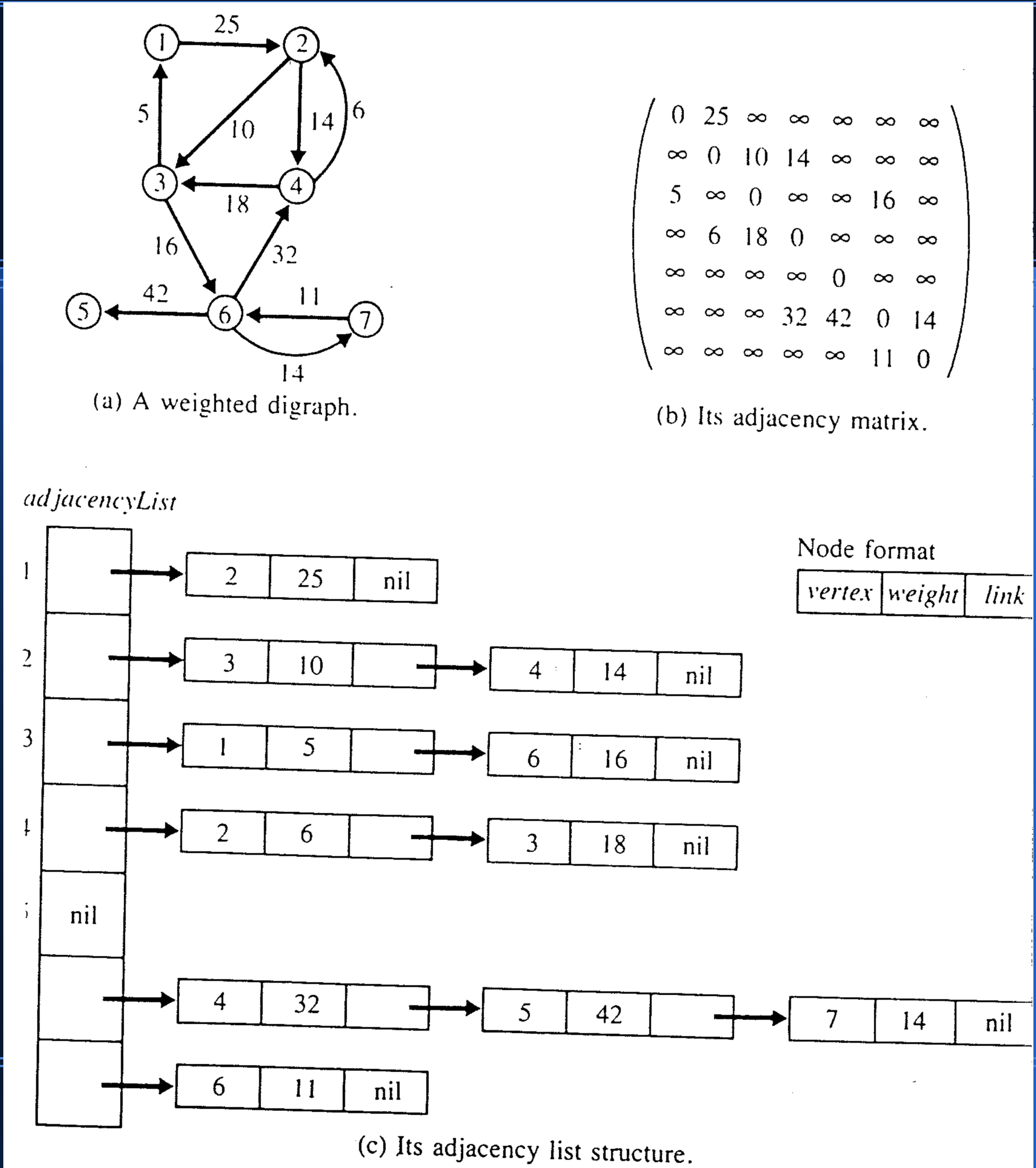
(c) Its adjacency list structure.

Figure 4.10 Representations for a graph.

Graphs and Digraphs



```
class DiGraph {

private int         size;    //number of vertices
private LIST[] AdjLists; //array of adjacency lists
private boolean[]   mark;    //to mark "visited"
```

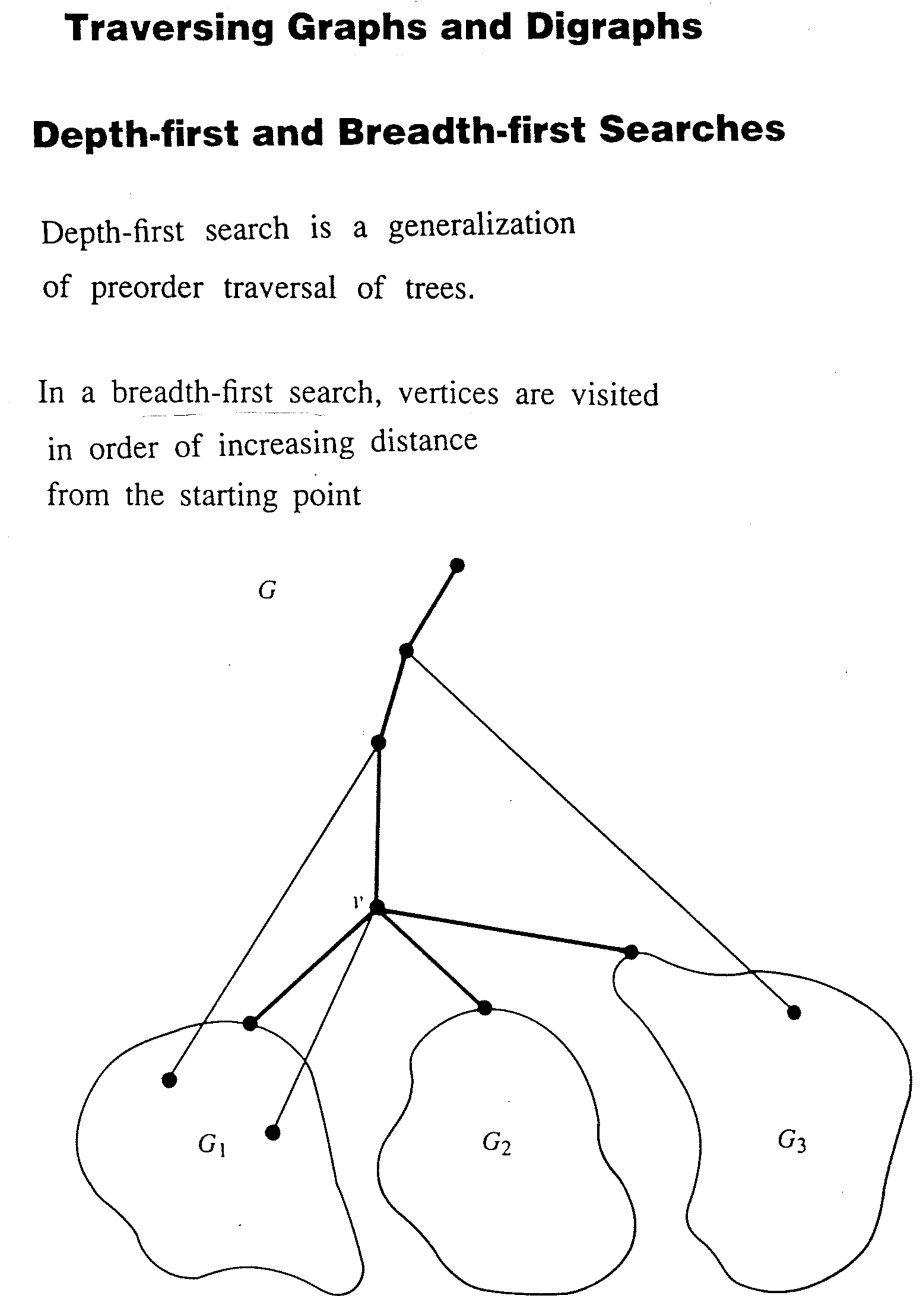


(a) A weighted digraph.

$$
\begin{pmatrix}
0 & 25 & \infty & \infty & \infty & \infty & \infty \\
\infty & 0 & 10 & 14 & \infty & \infty & \infty \\
5 & \infty & 0 & \infty & \infty & 16 & \infty \\
\infty & 6 & 18 & 0 & \infty & \infty & \infty \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & \infty & \infty & 32 & 42 & 0 & 14 \\
\infty & \infty & \infty & \infty & \infty & 11 & 0
\end{pmatrix}
$$

(b) Its adjacency matrix.

(c) Its adjacency list structure.



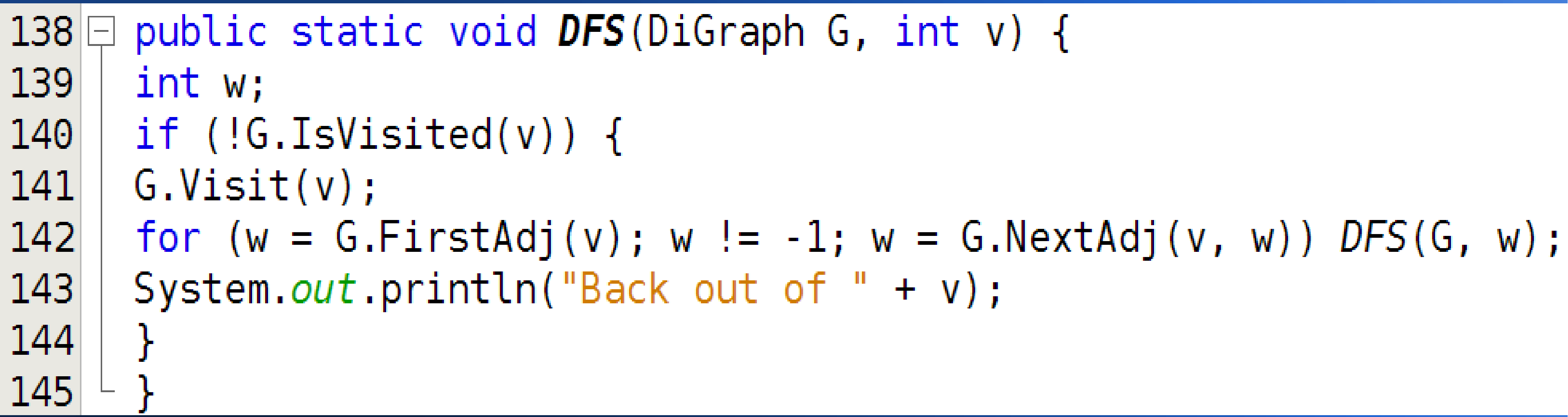
Traversing Graphs and Digraphs

Depth-first and Breadth-first Searches

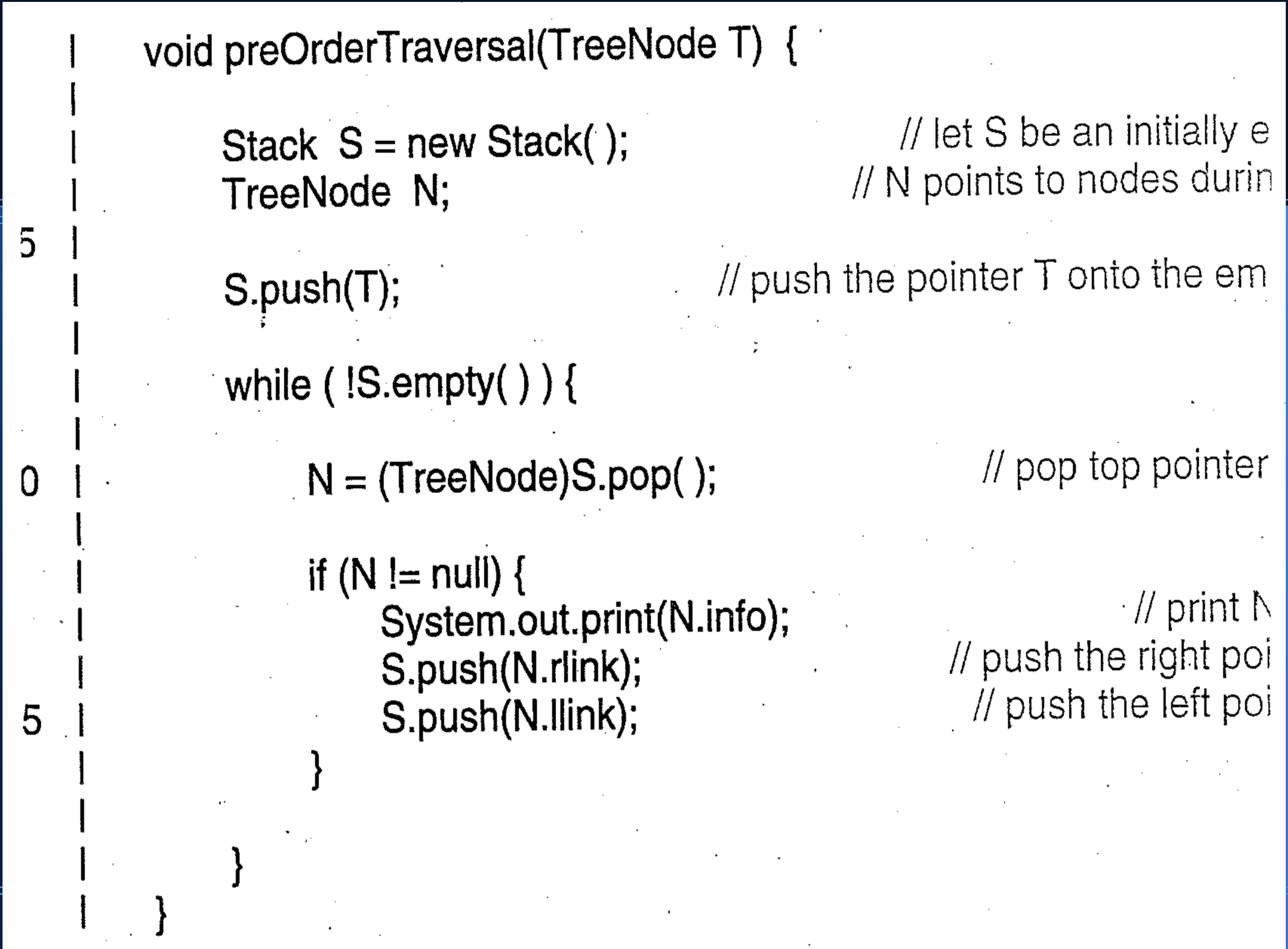
Depth-first search is a generalization of preorder traversal of trees.

In a breadth-first search, vertices are visited in order of increasing distance from the starting point

Graphs and Digraphs

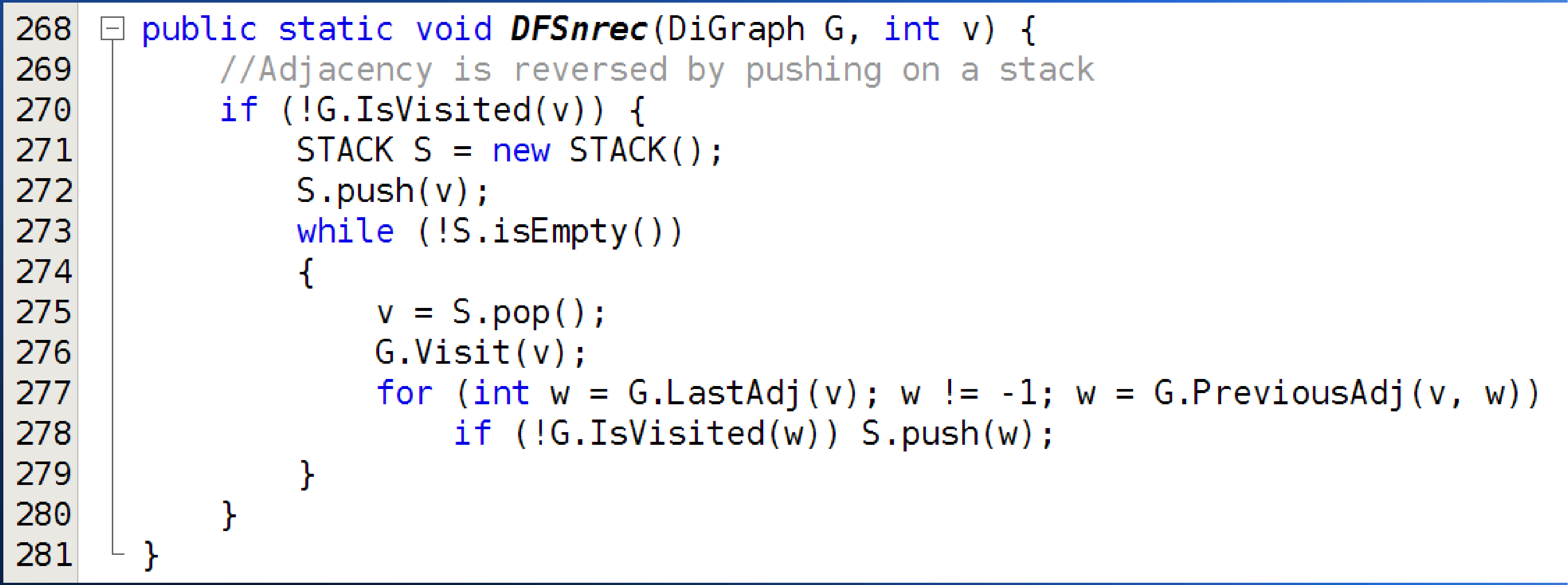


```
public static void DFS(DiGraph G, int v) {
int w;
if (!G.IsVisited(v)) {
G.Visit(v);
for (w = G.FirstAdj(v); w != -1; w = G.NextAdj(v, w)) DFS(G, w);
System.out.println("Back out of " + v);
}
}
```

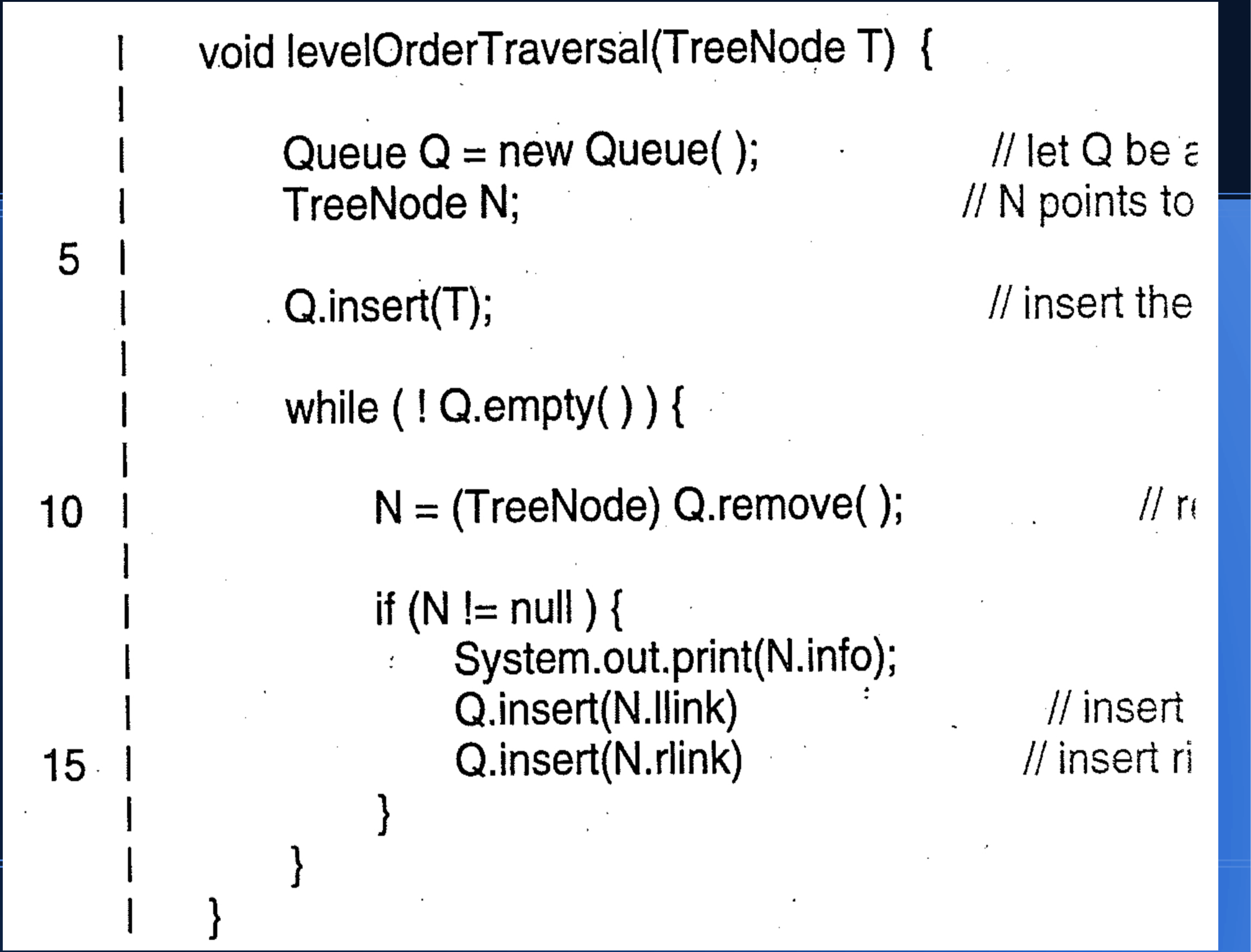


```
void preOrderTraversal(TreeNode T)  {

        Stack  S = new Stack( );                        // let S be an initially e
        TreeNode  N;                                  // N points to nodes durin

        S.push(T);                          // push the pointer T onto the em

        while ( !S.empty( ) ) {

                N = (TreeNode)S.pop( );                      // pop top pointer

                if (N != null) {
                        System.out.print(N.info);                   // print N
                        S.push(N.rlink);                   // push the right poi
                        S.push(N.llink);                    // push the left poi
                }

        }
}
```

Graphs and Digraphs

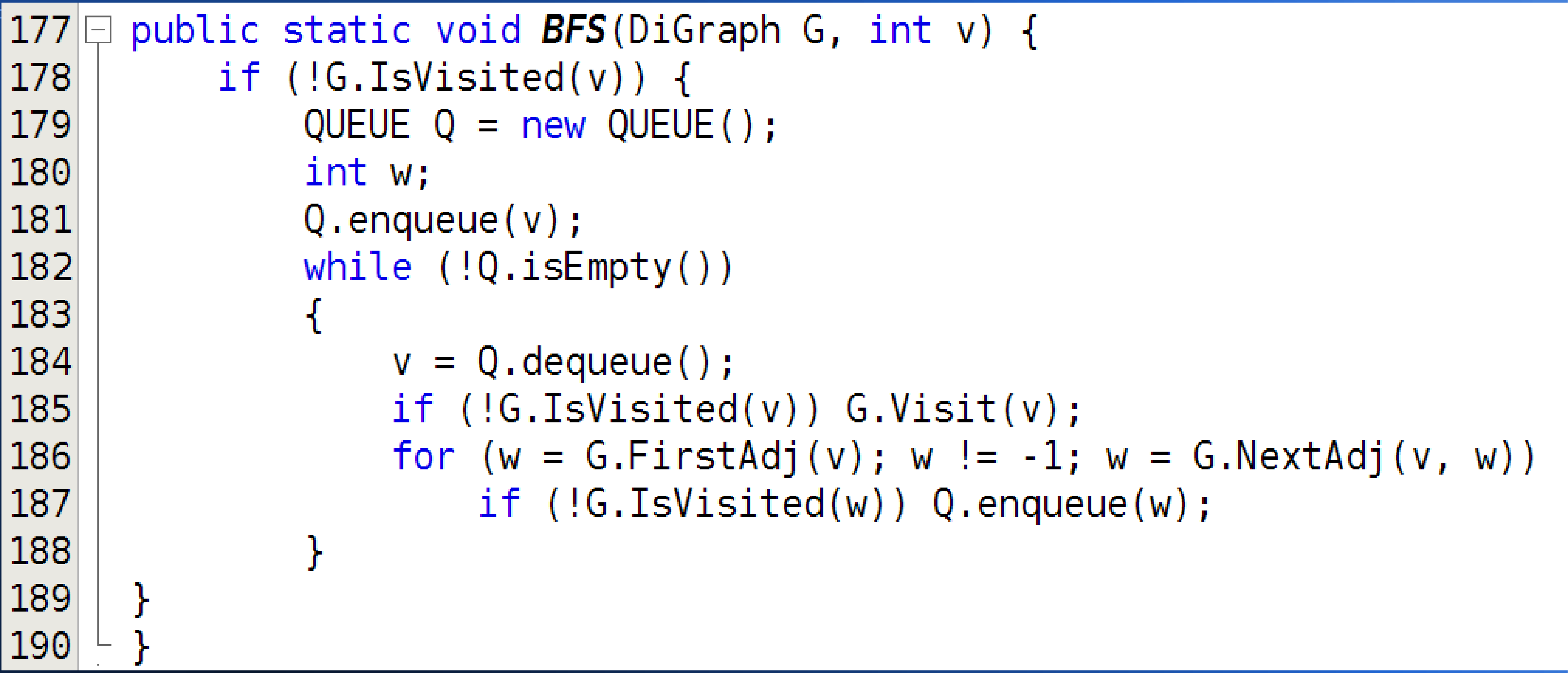


```java
public static void DFSnrec(DiGraph G, int v) {
    //Adjacency is reversed by pushing on a stack
    if (!G.IsVisited(v)) {
        STACK S = new STACK();
        S.push(v);
        while (!S.isEmpty())
        {
            v = S.pop();
            G.Visit(v);
            for (int w = G.LastAdj(v); w != -1; w = G.PreviousAdj(v, w))
                if (!G.IsVisited(w)) S.push(w);
        }
    }
}
```



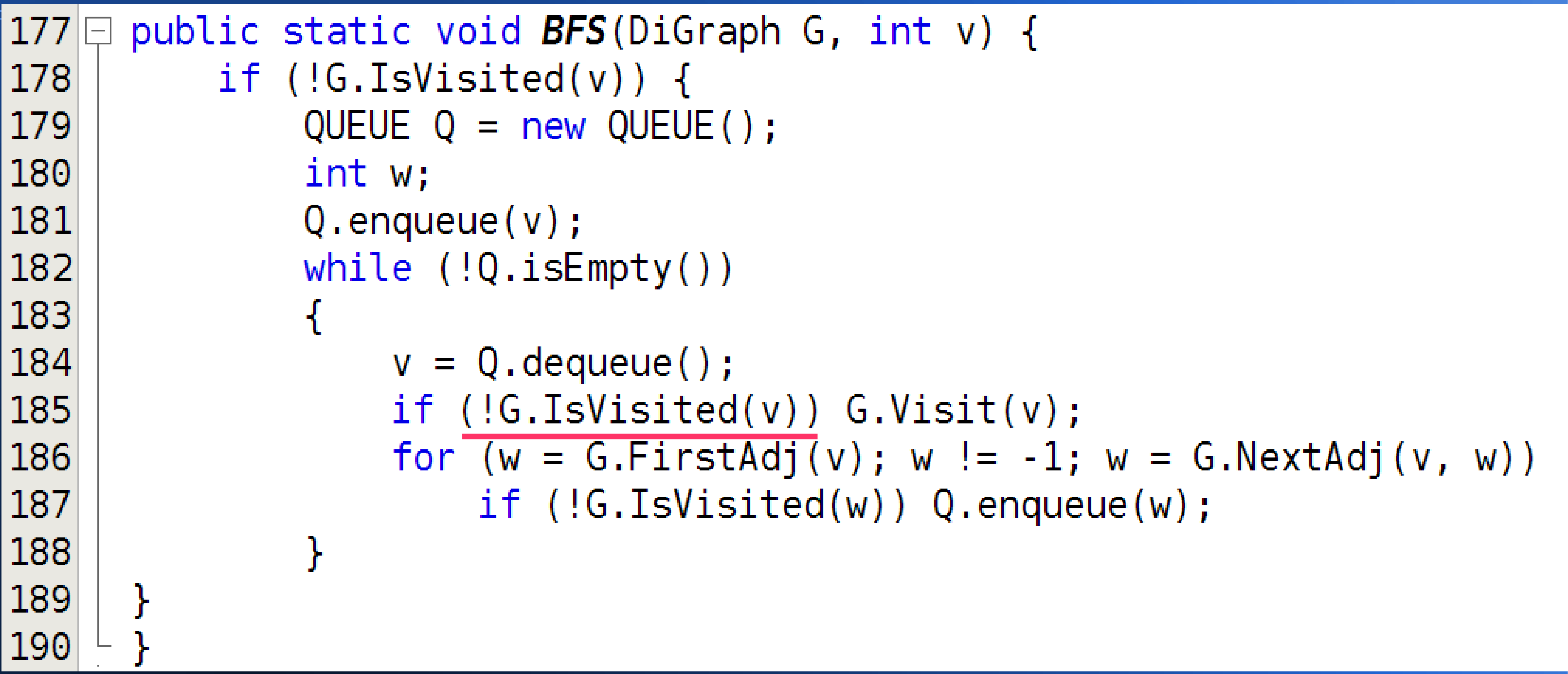
```
void levelOrderTraversal(TreeNode T)  {

      Queue Q = new Queue( );                      // let Q be a
      TreeNode N;                                 // N points to

      Q.insert(T);                                 // insert the

      while ( ! Q.empty( ) ) {

            N = (TreeNode) Q.remove( );                  // r

            if (N != null ) {
                  System.out.print(N.info);
                  Q.insert(N.llink)                     // insert
                  Q.insert(N.rlink)                    // insert ri
            }
      }
}
```

Graphs and Digraphs



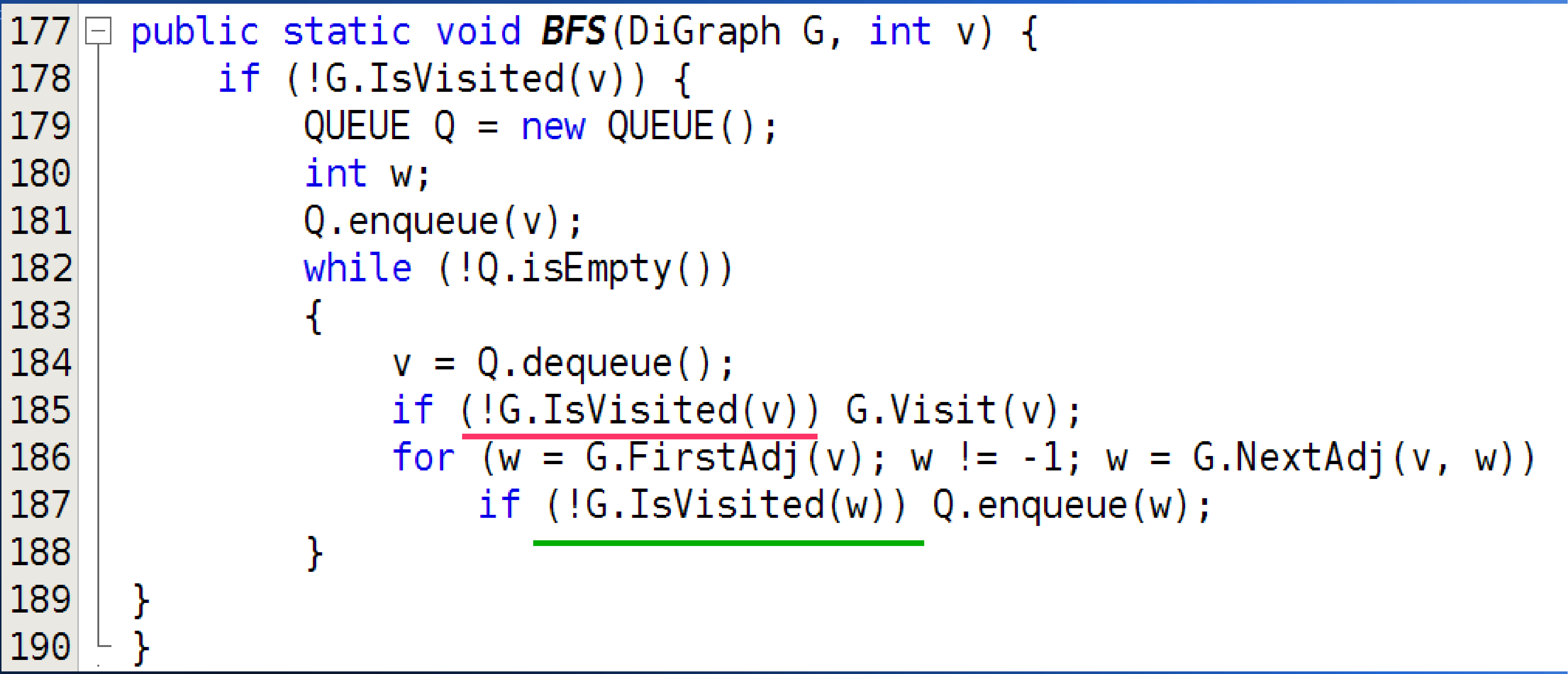
```
public static void BFS(DiGraph G, int v) {
    if (!G.IsVisited(v)) {
        QUEUE Q = new QUEUE();
        int w;
        Q.enqueue(v);
        while (!Q.isEmpty())
        {
            v = Q.dequeue();
            if (!G.IsVisited(v)) G.Visit(v);
            for (w = G.FirstAdj(v); w != -1; w = G.NextAdj(v, w))
                if (!G.IsVisited(w)) Q.enqueue(w);
        }
}
}
```

Graphs and Digraphs

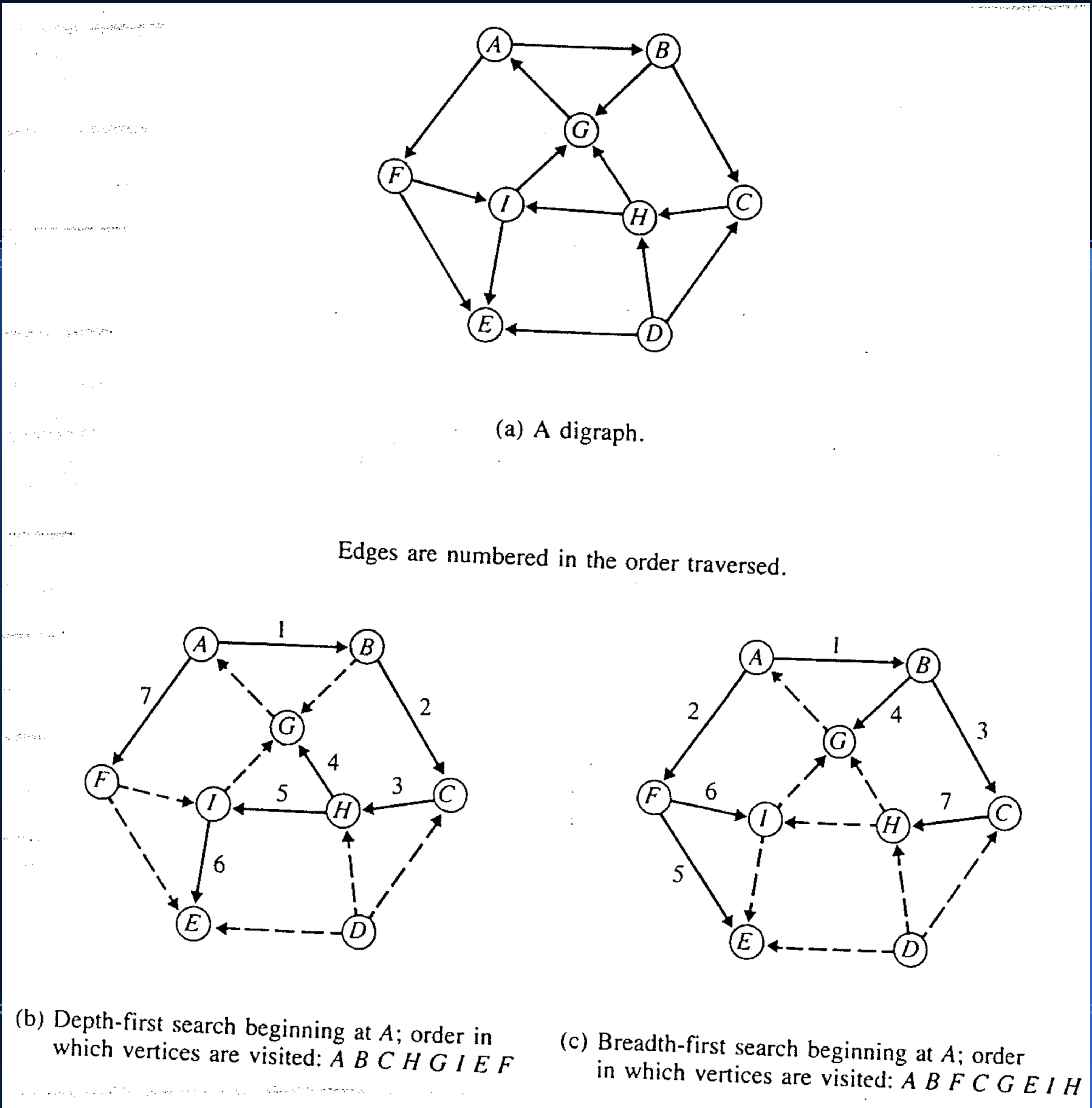


```java
public static void BFS(DiGraph G, int v) {
    if (!G.IsVisited(v)) {
        QUEUE Q = new QUEUE();
        int w;
        Q.enqueue(v);
        while (!Q.isEmpty())
        {
            v = Q.dequeue();
            if (!G.IsVisited(v)) G.Visit(v);
            for (w = G.FirstAdj(v); w != -1; w = G.NextAdj(v, w))
                if (!G.IsVisited(w)) Q.enqueue(w);
        }
}
}
```

Graphs and Digraphs



```
public static void BFS(DiGraph G, int v) {
    if (!G.IsVisited(v)) {
        QUEUE Q = new QUEUE();
        int w;
        Q.enqueue(v);
        while (!Q.isEmpty())
        {
            v = Q.dequeue();
            if (!G.IsVisited(v)) G.Visit(v);
            for (w = G.FirstAdj(v); w != -1; w = G.NextAdj(v, w))
                if (!G.IsVisited(w)) Q.enqueue(w);
        }
    }
}
```

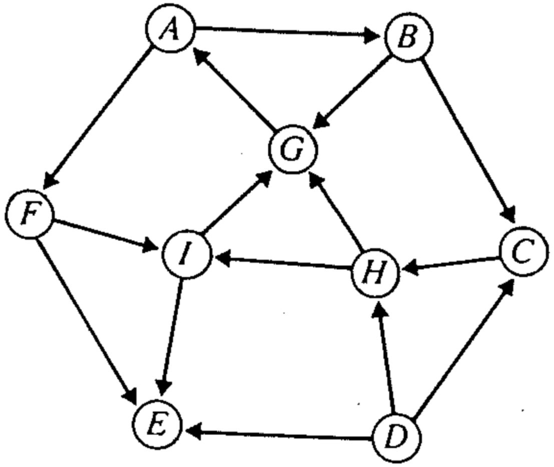


(a) A digraph.

Edges are numbered in the order traversed.

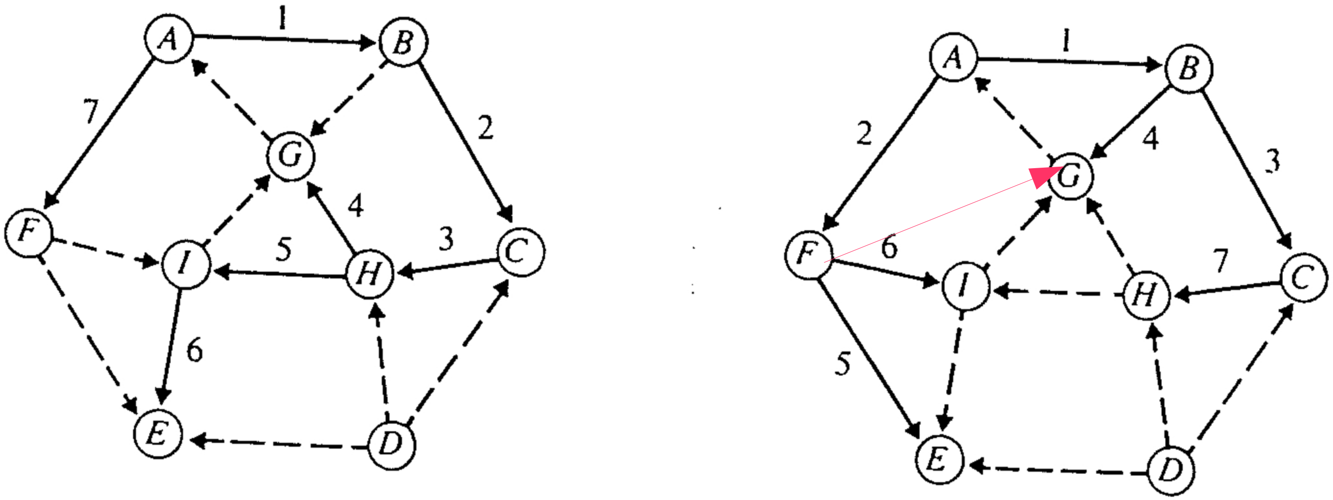
(b) Depth-first search beginning at *A*; order in which vertices are visited: *A B C H G I E F*

(c) Breadth-first search beginning at *A*; order in which vertices are visited: *A B F C G E I H*



(a) A digraph.

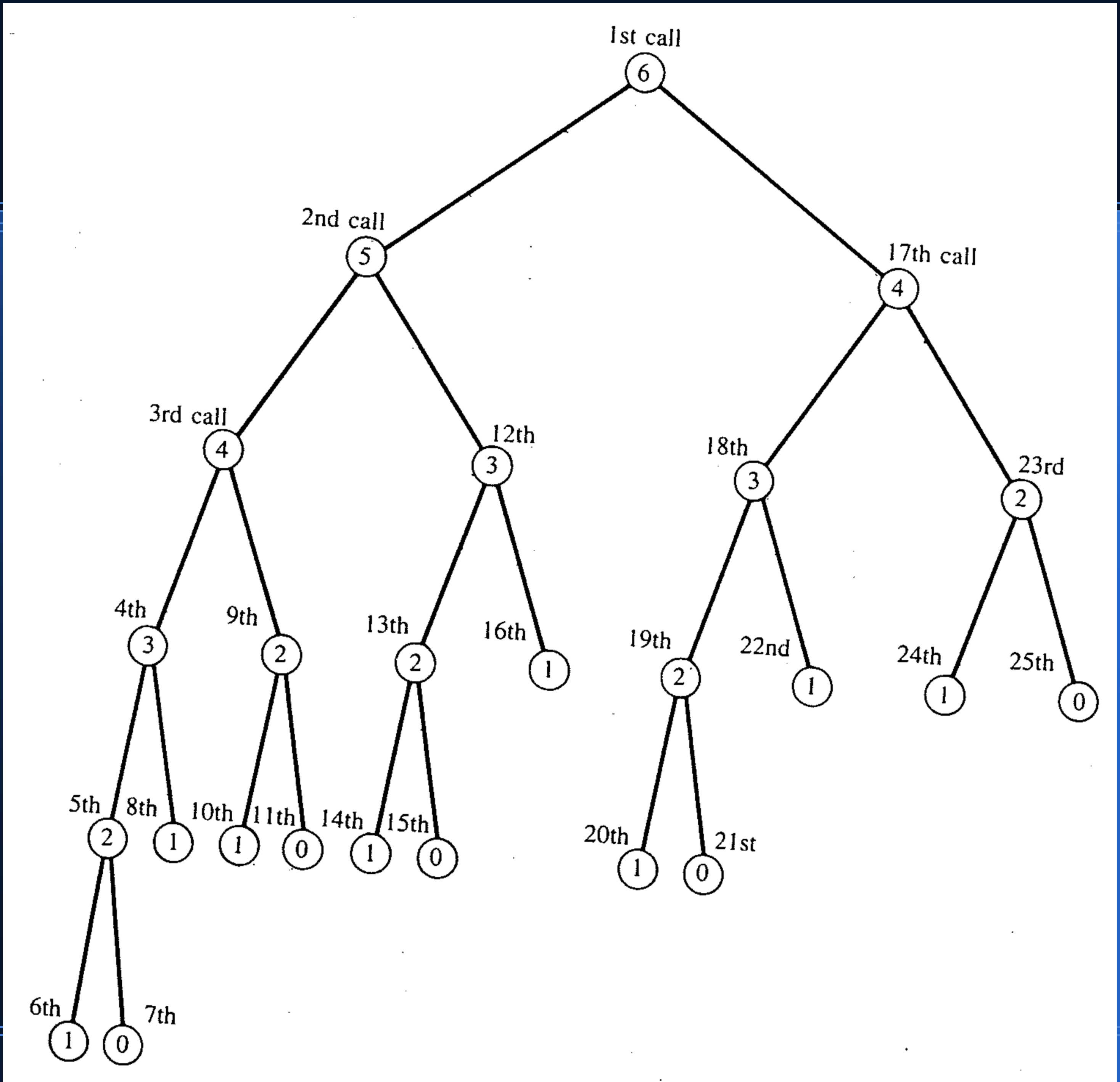
Edges are numbered in the order traversed.

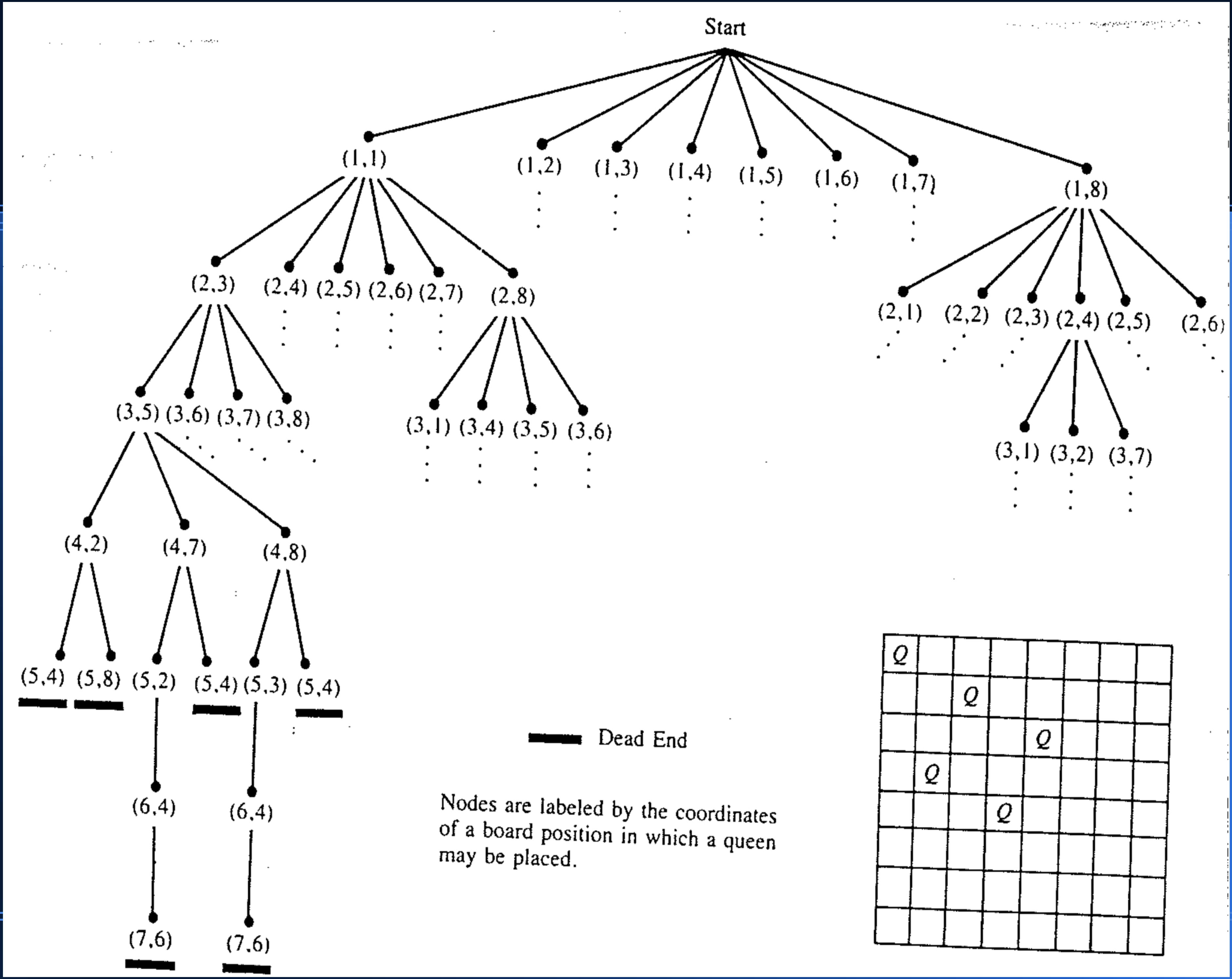


Variant

(b) Depth-first search beginning at *A*; order in which vertices are visited: *A B C H G I E F*

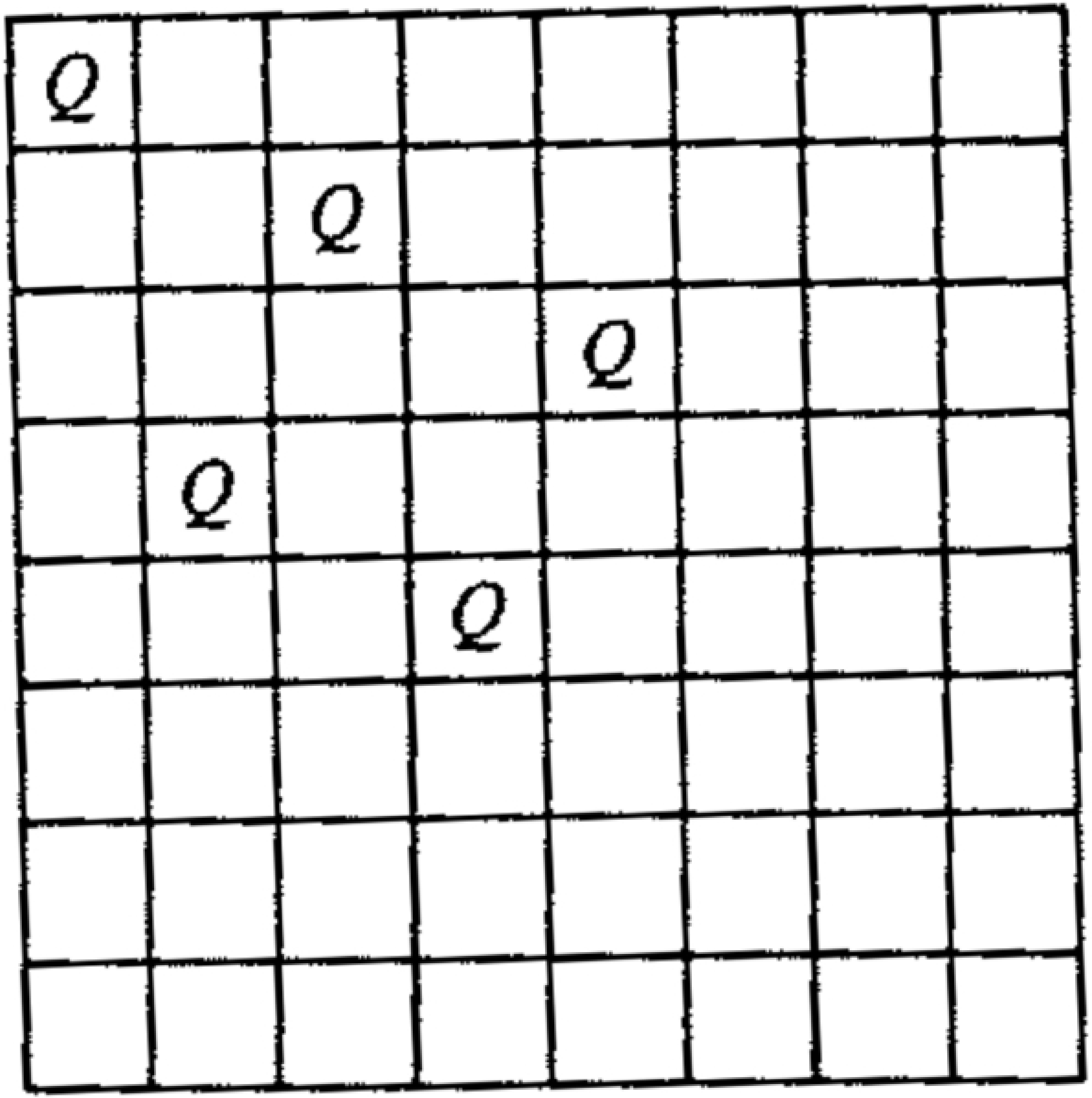
(c) Breadth-first search beginning at *A*; order in which vertices are visited: *A B F C G E I H*


1st call
6
2nd call
5
17th call
4
3rd call
4
12th
3
18th
3
23rd
2
4th
3
9th
2
13th
2
16th
1
19th
2
22nd
1
24th
1
25th
0
5th
2
8th
1
10th
1
11th
0
14th
1
15th
0
20th
1
21st
0
6th
1
7th
0

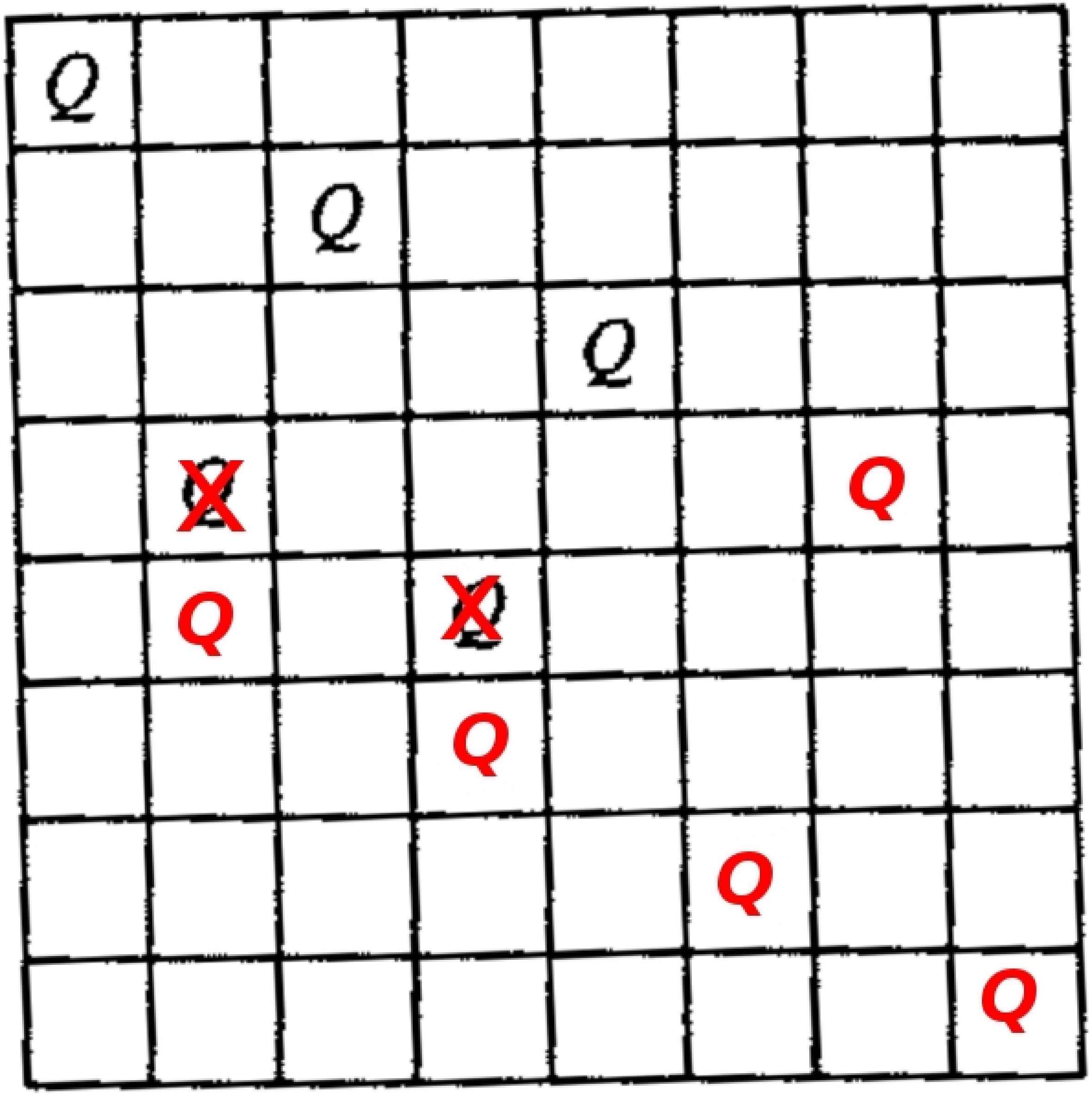


Nodes are labeled by the coordinates of a board position in which a queen may be placed.

Q							
		Q					
				Q			
	Q						
			Q				

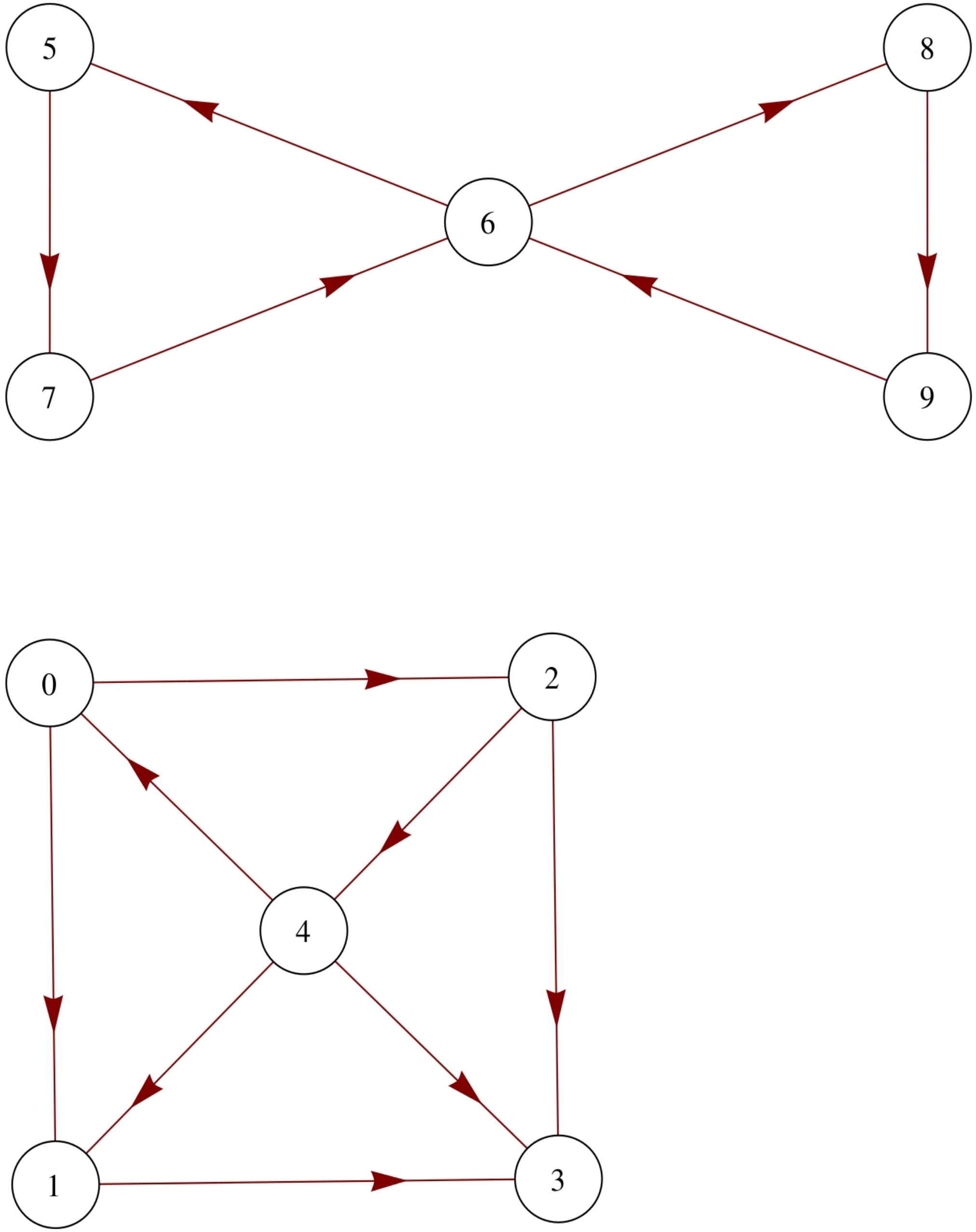


Q							
		Q					
				Q			
	Q						
			Q				

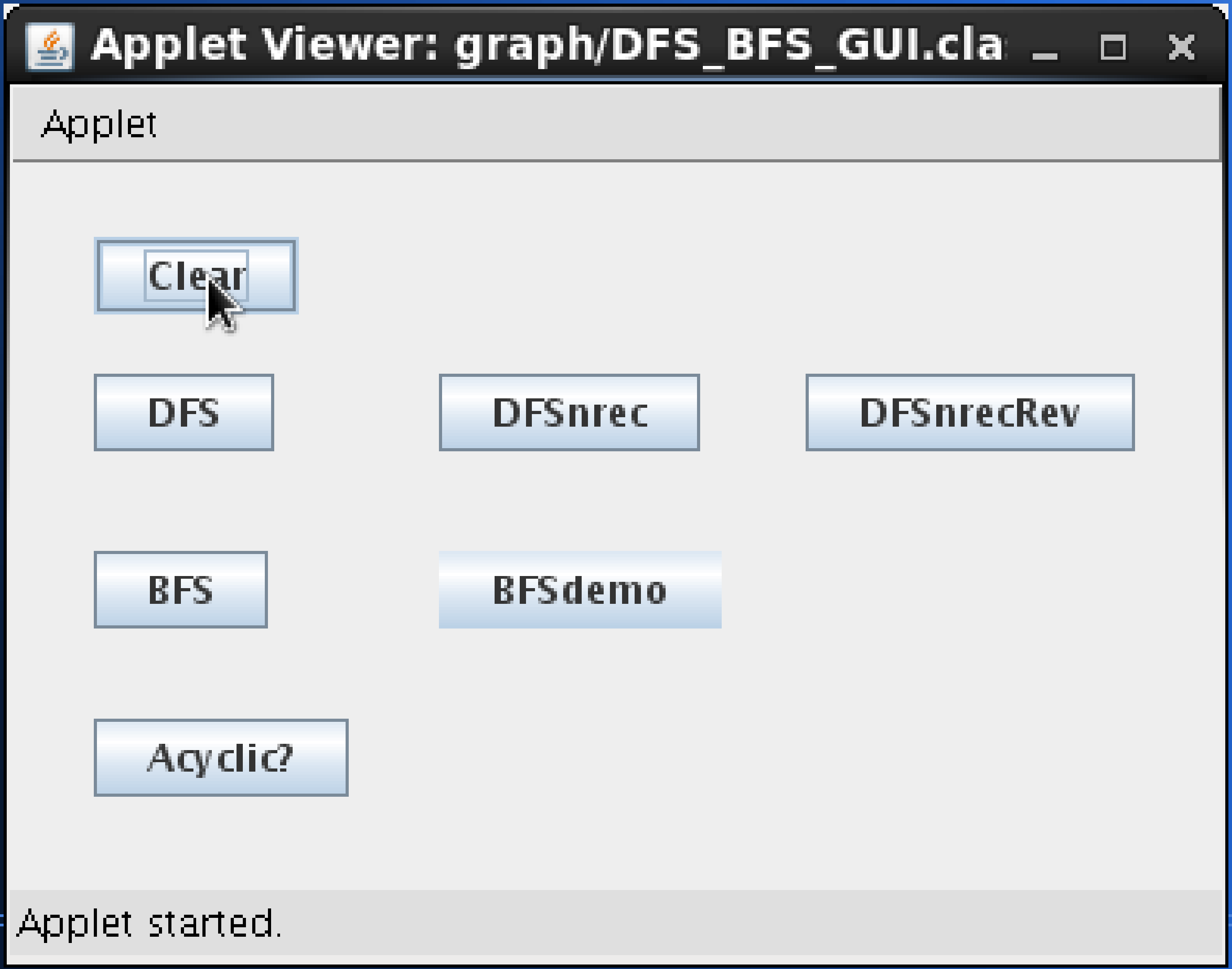


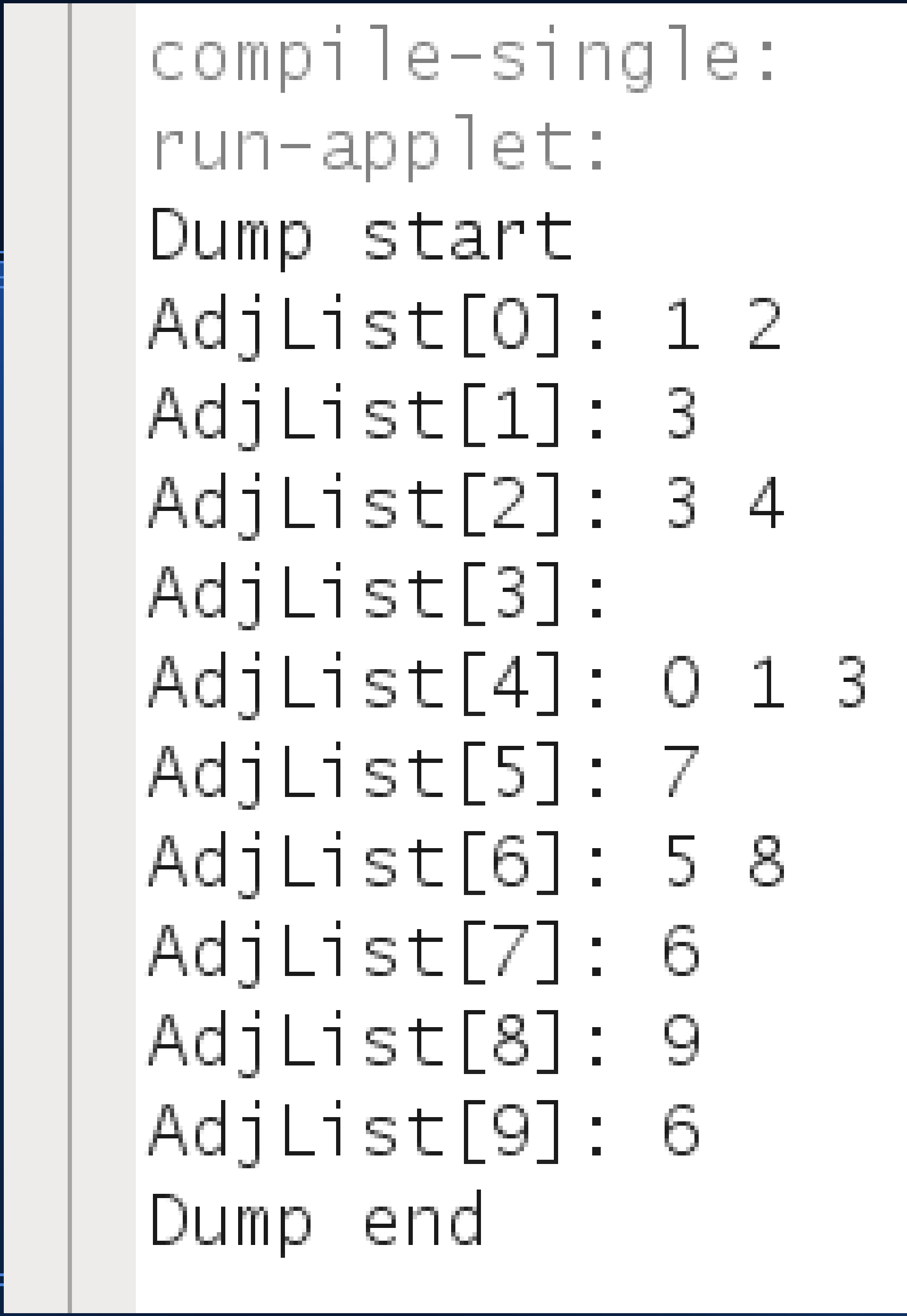
Q							
		Q					
				Q			
	~~Q~~					Q	
	Q		~~Q~~				
			Q				
					Q		
							Q

Demo of Graph progr. in java

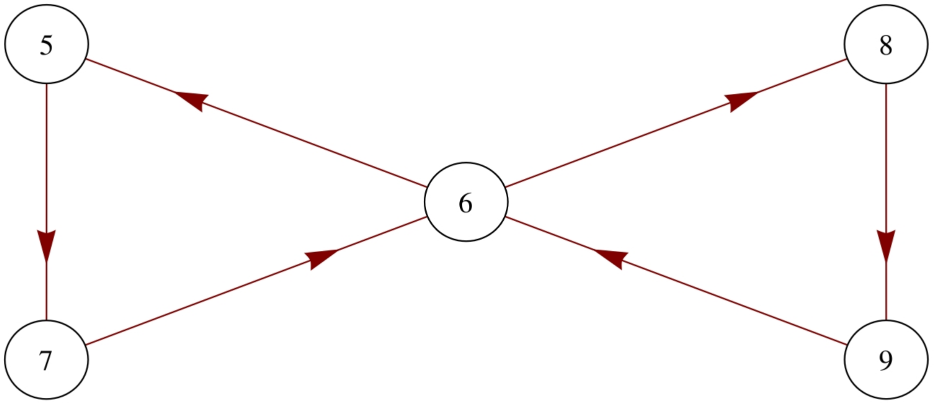


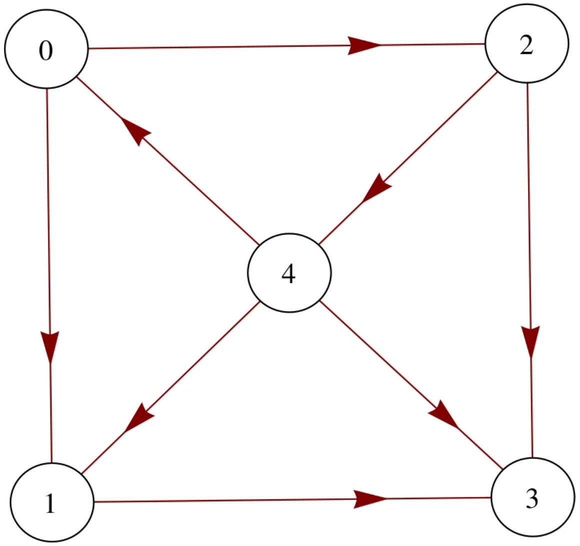
DFS & BFS


Applet Viewer: graph/DFS_BFS_GUI.cla
Applet
Clear
DFS
DFSnrec
DFSnrecRev
BFS
BFSdemo
Acyclic?
Applet started.

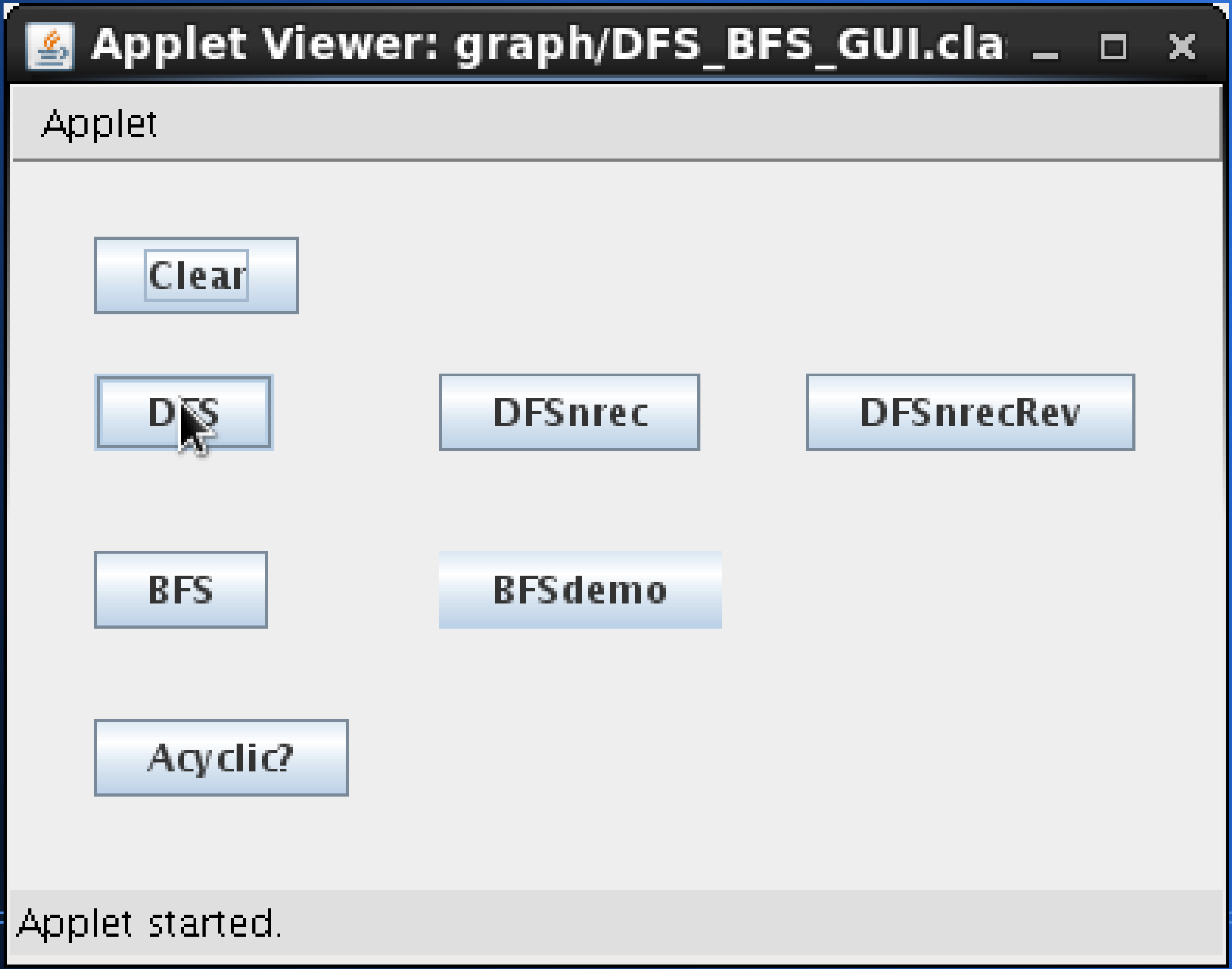


```
compile-single:
run-applet:
Dump start
AdjList[0]: 1 2
AdjList[1]: 3
AdjList[2]: 3 4
AdjList[3]:
AdjList[4]: 0 1 3
AdjList[5]: 7
AdjList[6]: 5 8
AdjList[7]: 6
AdjList[8]: 9
AdjList[9]: 6
Dump end
```



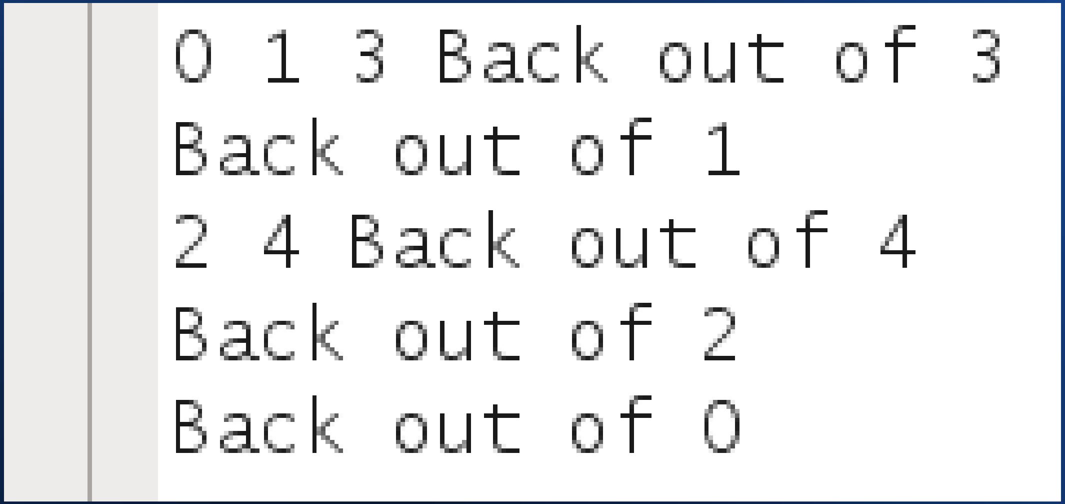


DFS & BFS

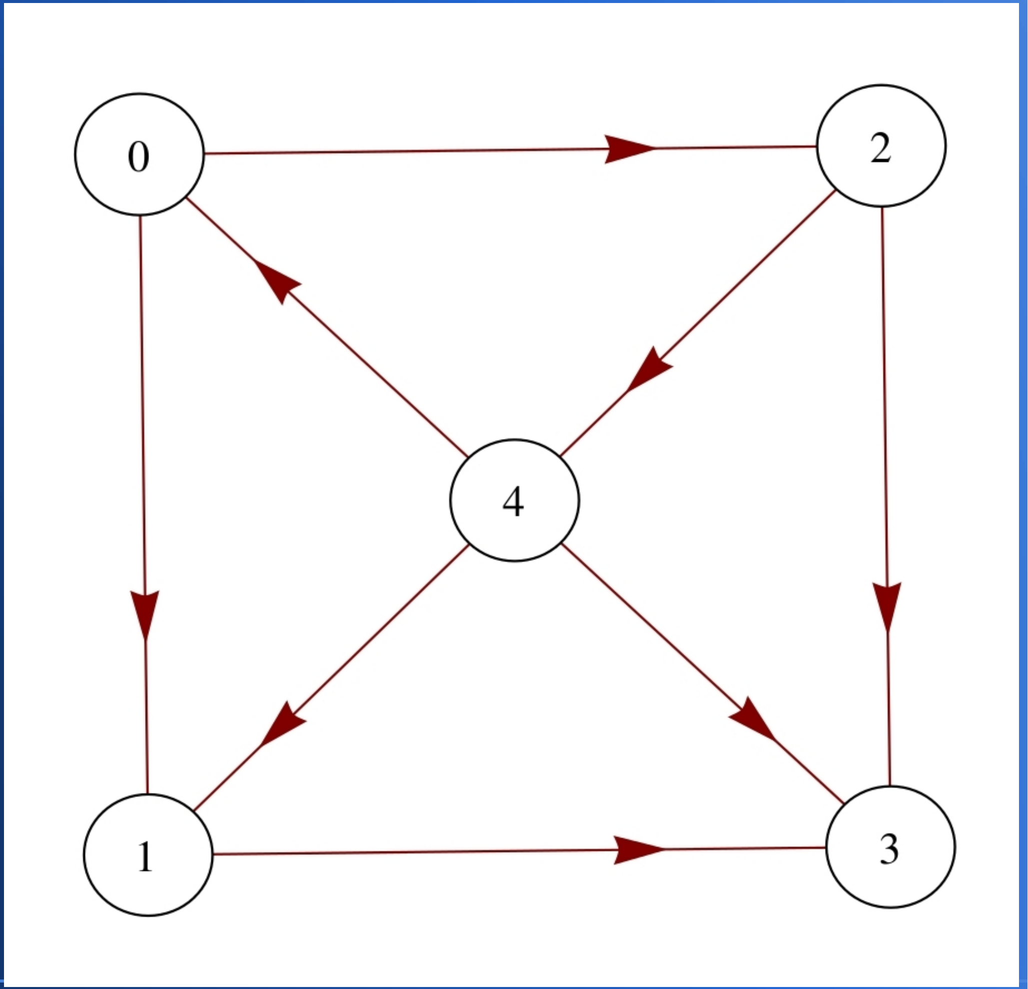


Applet started.

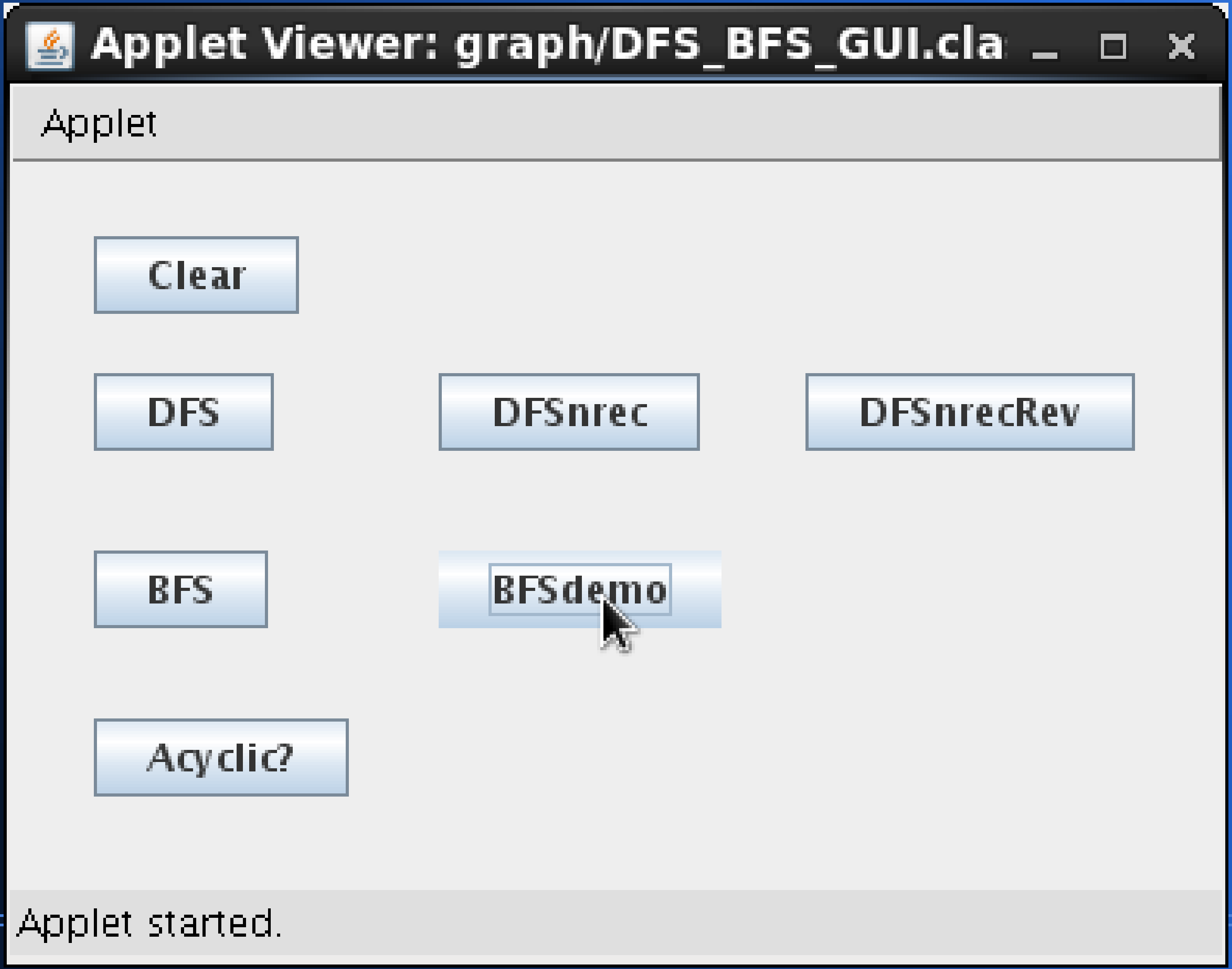
DFS & BFS



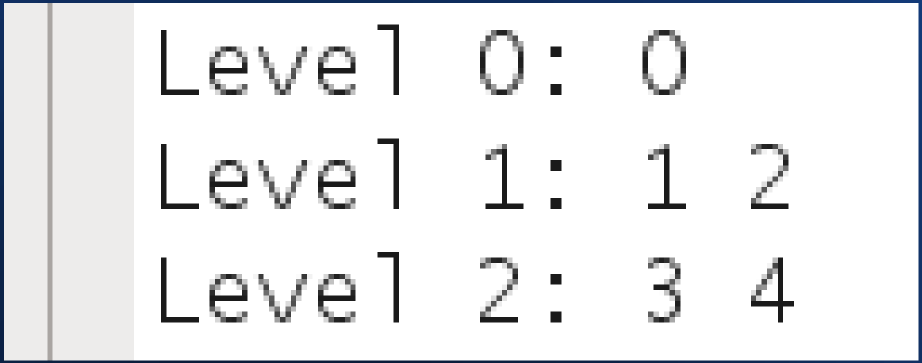
```
0 1 3 Back out of 3
Back out of 1
2 4 Back out of 4
Back out of 2
Back out of 0
```



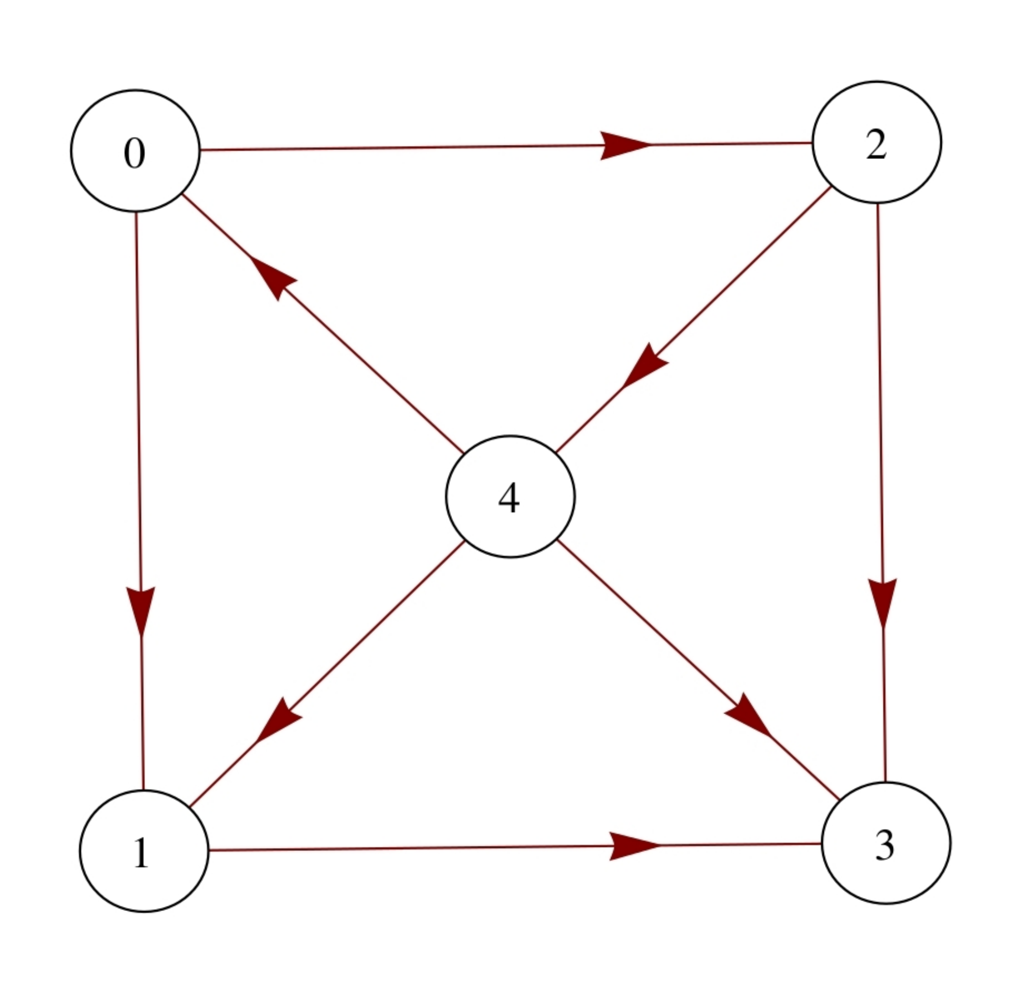
DFS & BFS


Applet Viewer: graph/DFS_BFS_GUI.cla
Applet
Clear
DFS
DFSnrec
DFSnrecRev
BFS
BFSdemo
Acyclic?
Applet started.

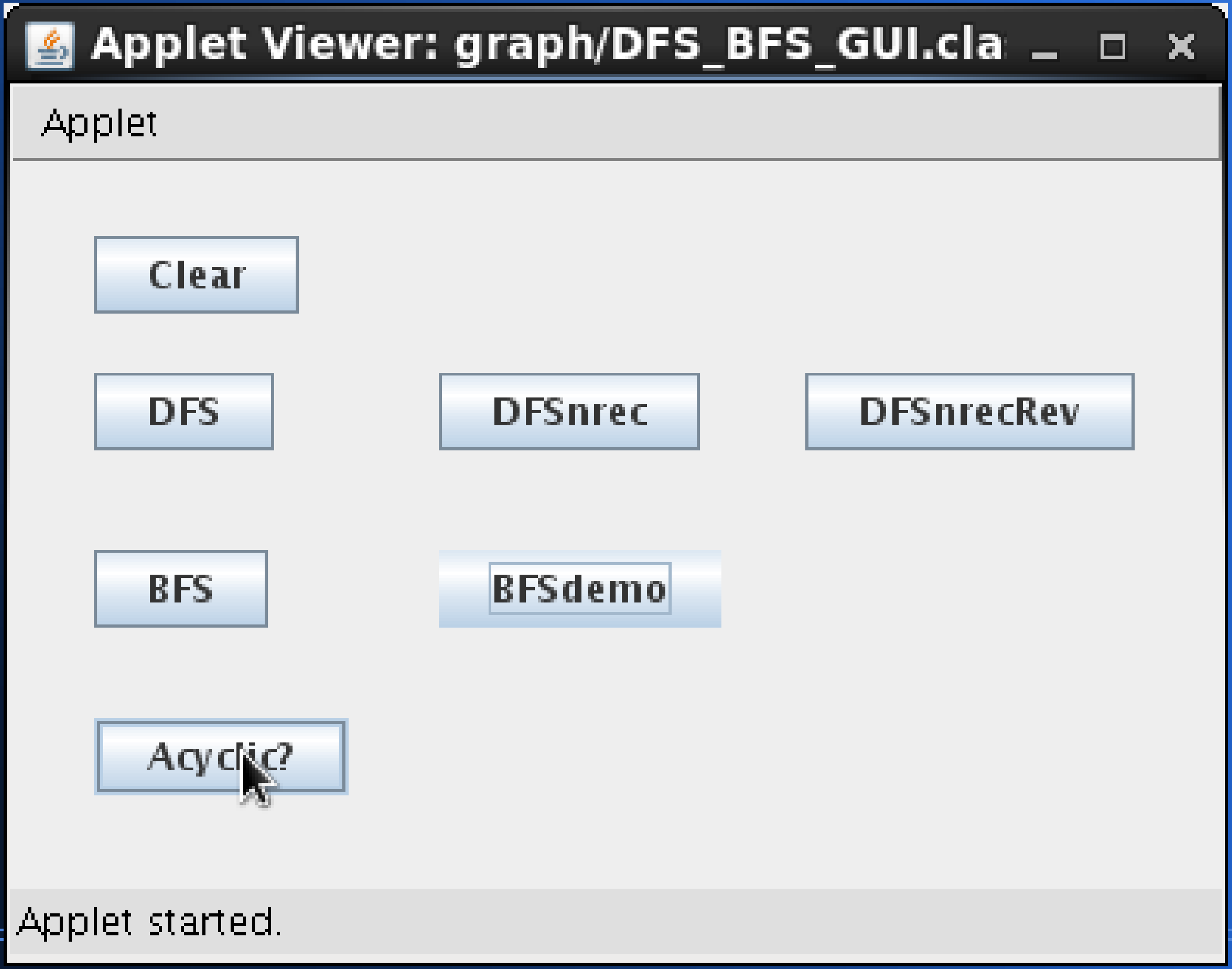
DFS & BFS



```
Level 0: 0
Level 1: 1 2
Level 2: 3 4
```

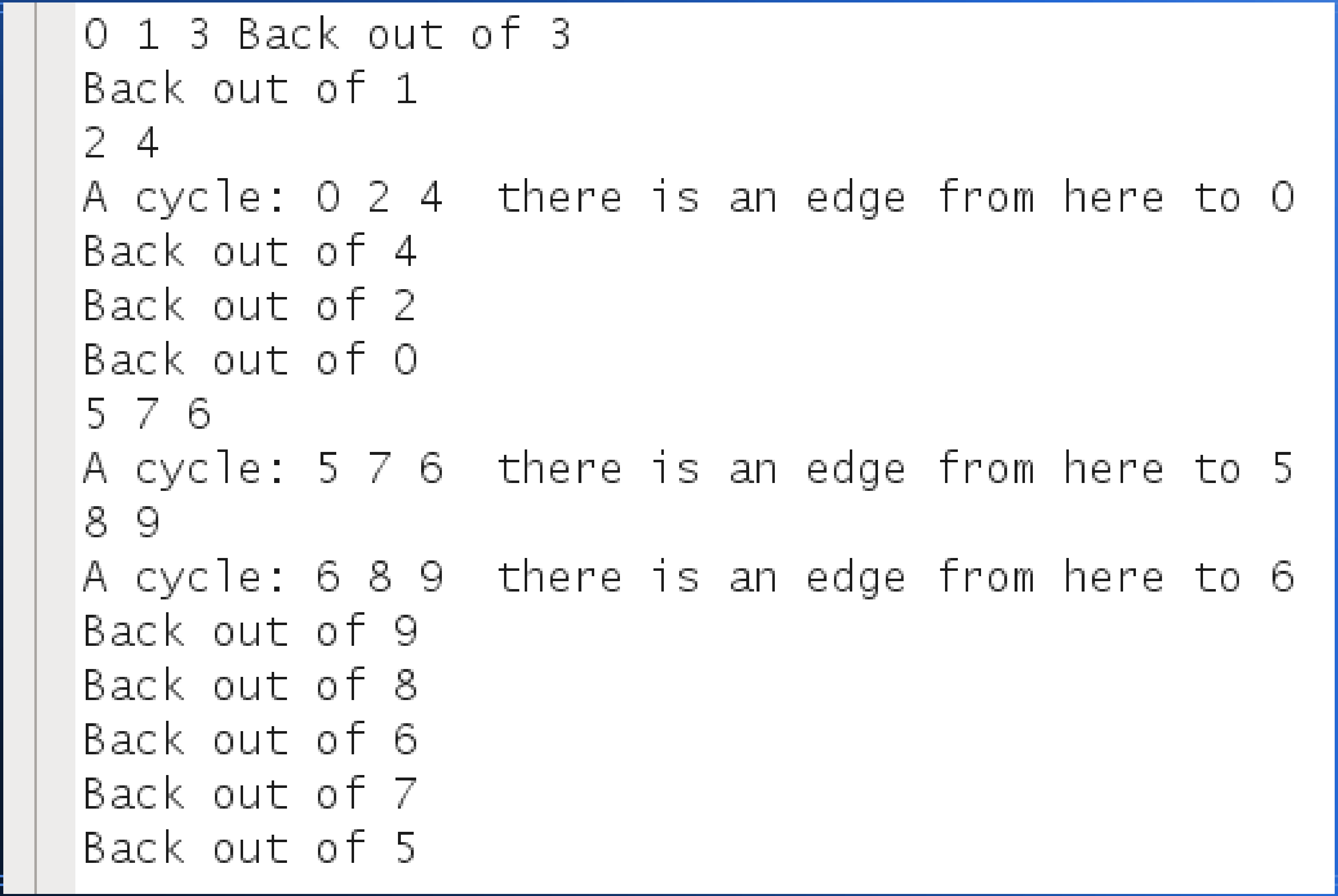


DFS & BFS



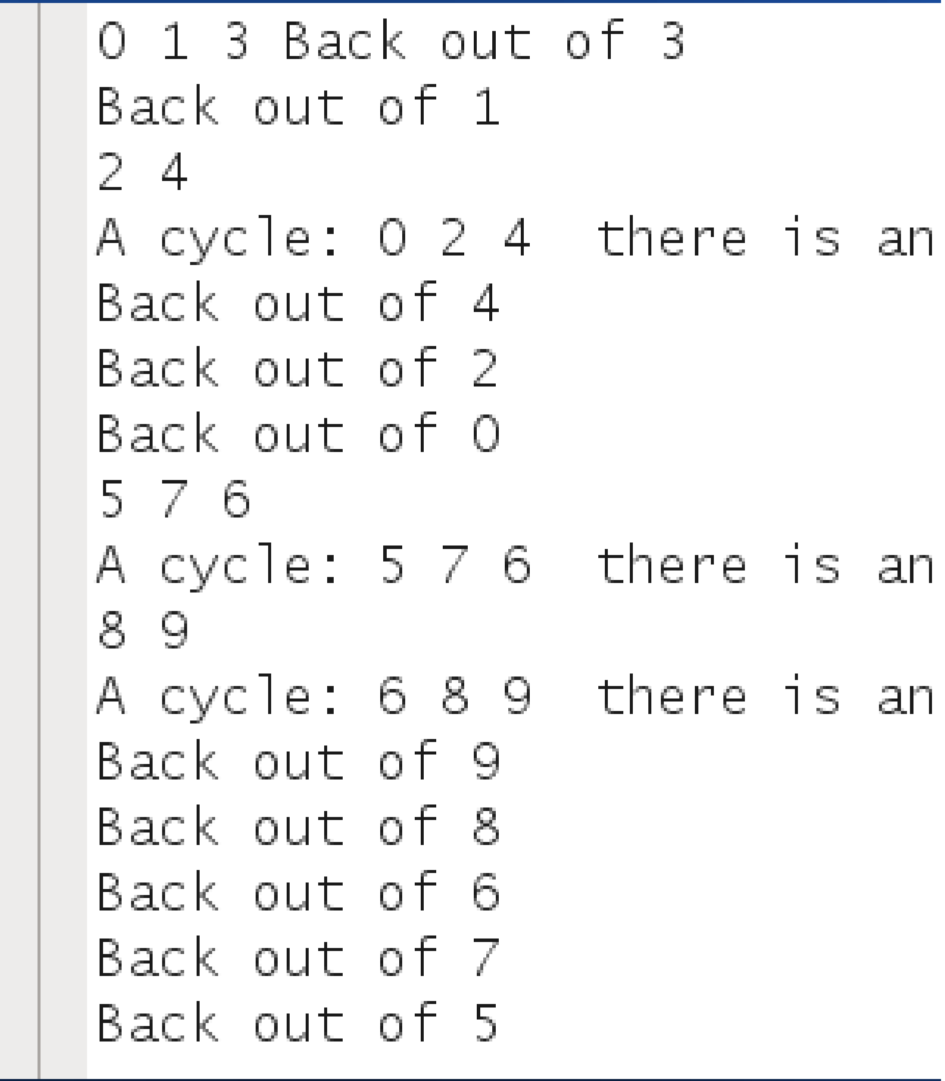
Applet started.

DFS & BFS

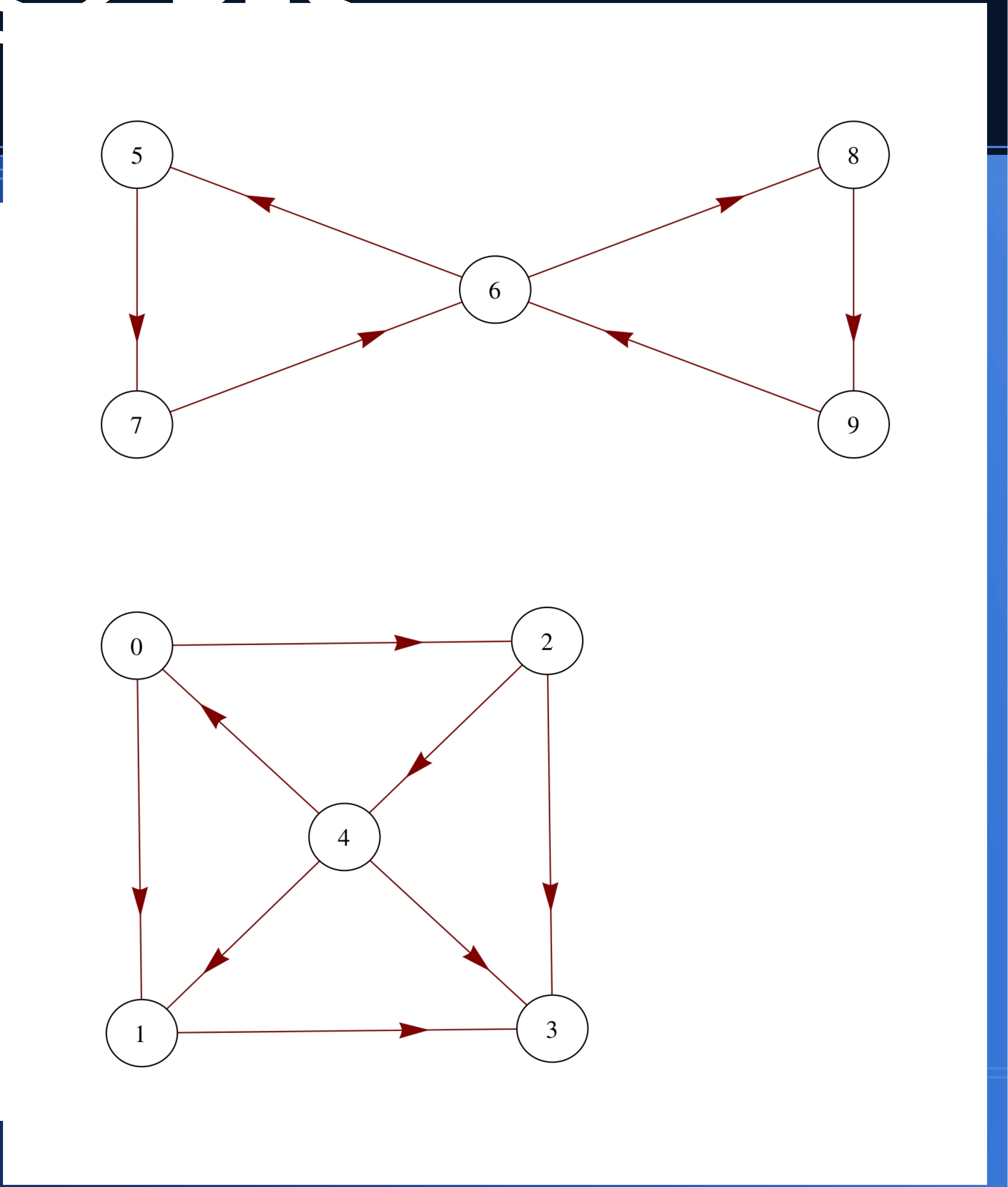


```
0 1 3 Back out of 3
Back out of 1
2 4
A cycle: 0 2 4  there is an edge from here to 0
Back out of 4
Back out of 2
Back out of 0
5 7 6
A cycle: 5 7 6  there is an edge from here to 5
8 9
A cycle: 6 8 9  there is an edge from here to 6
Back out of 9
Back out of 8
Back out of 6
Back out of 7
Back out of 5
```

DFS & BFS



```
0 1 3 Back out of 3
Back out of 1
2 4
A cycle: 0 2 4  there is an
Back out of 4
Back out of 2
Back out of 0
5 7 6
A cycle: 5 7 6  there is an
8 9
A cycle: 6 8 9  there is an
Back out of 9
Back out of 8
Back out of 6
Back out of 7
Back out of 5
```



This all will be covered on Final