Graph ADT and Basic Graph Search Algorithms

1. Complete the following table of properties of the graph with True/False

Property	True/False
Directed	**Y**
Cyclic	**Y**
Connected	**N e.g. no path from 6 to 3**
Weighted	**N**

2. For node 2 in Figure 1,
- What is the in-degree? **2**
- What is the out-degree? **2**
- What nodes are adjacent to 2? **3, 5**
- What nodes are adjacent from 2? **1,5**
- Give a paths to node 6, **2,5,6**
- Are there any more paths to node 6?
  Yes an infinite number, e.g. 2,5,2,5,6 or 2,5,2,5,2,5,6.

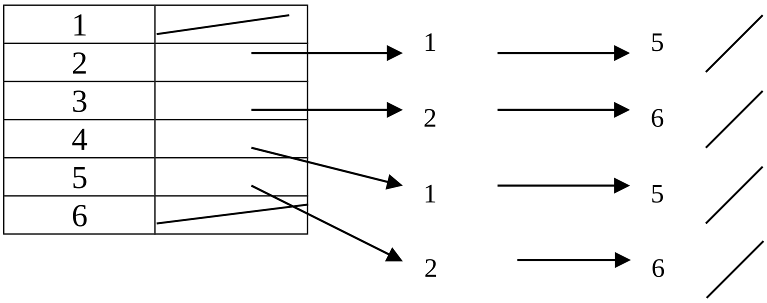
3. Draw an adjacency matrix representation of the graph in Figure 1.

	1	**2**	**3**	**4**	**5**	**6**
1	F	F	F	F	F	F
2	T	F	F	F	T	F
3	F	T	F	F	F	T
4	T	F	F	F	T	F
5	F	T	F	F	F	T
6	F	F	F	F	F	F

4. Draw an adjacency list representation of the graph in Figure 1.

Either of the following representations is acceptable:

1 []
2 [1,5]
3 [2,6]
4 [1,5]
5 [2,6]
6 []



5. Write a method `boolean existsEdge(int i, int j)` for the Adjacency Matrix Digraph (`AdjacencyDigraph`) class in the notes.

```
public boolean existsEdge(int i, int j)
   {
      if (i < 1 || j < 1 || i > n || j > n)
         throw new IllegalArgumentException("no vertex " + i +
“ or “ + j);
      else
         return a[i][j];
   }
```

6. Write the `int outDegree(int i)` and `int inDegree(int i)` methods for the Adjacency List Digraph (`LinkedDigraph`) class in the notes.

```
public int outDegree(int i)
  {
    if (i < 1 || i > n)
      throw new IllegalArgumentException("no vertex " + i);

    return aList[i].size();
  }

public int inDegree(int i)
  {
    if (i < 1 || i > n)
      throw new IllegalArgumentException("no vertex " + i);

    // count in edges at vertex i
    int sum = 0;
    for (int j = 1; j <= n; j++)
      if (aList[j].indexOf(new EdgeNode(i)) != -1)
          sum++;

    return sum;
  }
```