COMP103 Data Structures and Algorithms

Tutorial 13: Graph (Solution)

A. Graph Data Structure

1. List differences between tree and graph data structure are as shown in the table below.

[7 marks]

Criteria	Tree	Graph
Path		
Root node		
Loops		
Complexity		
Traversal techniques		
Number of edges		
Model type		

Answer

The differences between tree and graph data structures are listed in the table below.

Criteria	Tree	Graph
Path	Only one between two vertices.	More than one path is allowed.
Root node	It has exactly one root node.	Graph does not have a root node.
Loops	No loops are permitted.	Graph can have loops.
Complexity	Less complex	More complex comparatively
Traversal techniques	Pre-order, in-order and post-order.	Breadth-first search and depth-first search.
Number of edges	n-1 (where n is the number of nodes)	Not defined
Model type	Hierarchical	Network

2. What is graph? Devise an expression that describe graph. [2 marks]

Answer

Graph is a set of objects, called nodes/vertices (v), together with a collection of pairwise connections between them, called links/edges (e). The expression for graph is:

$$G = (v, e)$$

3. What are three types of graphs? Briefly describe their characteristics. [6 marks]

Answer

- Directed graphs - The edges that connect the vertices of the graph may be directed or undirected vs. an undirected graph, where the information about the direction of edges is not stored.
- Weighted graph - Each edge of a graph can be associated with a value that represents the weight of the edge. This property can be especially useful to determine the optimal path while traversing a graph.
- Cyclic graph - An acyclic graph is a directed graph, where at least one vertex ends up having a connection or relation with itself.

4. Classify graph according to its edge connections. [3 marks]

Answer

Edge connections:

- Directed graph: An edge (u, v) is directed from u to v if the pair is ordered, with u preceding v.
- Undirected graph: the pair (u, v) is not ordered.
- Mixed graph: contain both directed and unidirected graphs.

5. What is the number of edges present in a complete graph having n vertices? [2 marks]

Answer

Number of ways in which every vertex, n can be connected to each other is:

$$\text{edge} = \frac{n * (n-1)}{2}$$

6. Fill in the blanks given below for the terms used in the graph data structure. [3 marks]

   - A _____ in a graph is a list of nodes such that each node and its successor in the list are connected by an edge in the graph.
   - A _____ in a graph is a path whose first and last nodes are the same.
   - An _____ is one in which there is no cycle.

   Answer

   The keywords matching the blanks are shown below.

   - A path in a graph is a list of nodes such that each node and its successor in the list are connected by an edge in the graph.
   - A cycle in a graph is a path whose first and last nodes are the same.
   - An acyclic graph is one in which there is no cycle.

7. From the examples provided below, which one is classified as a weighted graph? [2 marks]

   - Road map connecting some cities in an island.
   - Social network of friendships among persons.
   - Relationships between tables in a database system.
   - Structural hierarchies of employees in a company.

   Answer:

   Roads between cities in the road map most often depict the relative distances between them.

8. How do you traverse graph? [2 marks]

   Answer

   There are two ways in which a graph can be traversed: breadth-first and depth-first. We will understand with the help of code how they both differ.

9. What is breadth-first traversal in the graph data structure? [2 marks]

   Answer

   Breadth-first traversal, as the name implies, means that all the nodes at a particular level on the graph are first visited, before proceeding down to the next level. So, it involves visiting all nodes

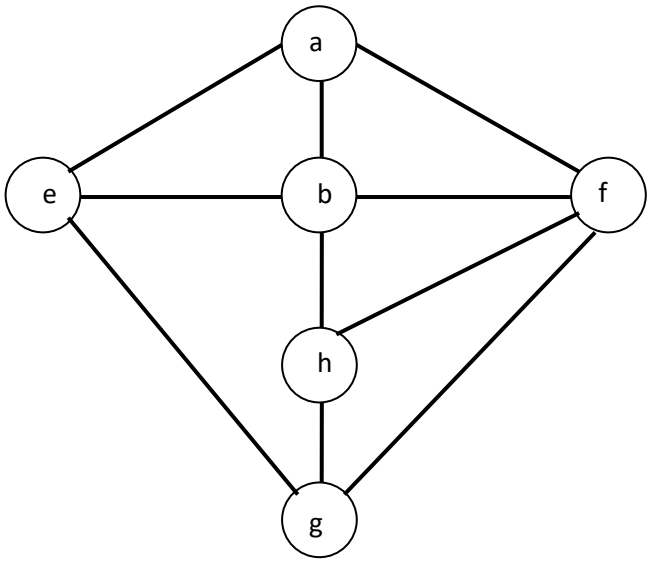
across the breadth of the graph before proceeding to go down.

10. What is depth-first traversal in the graph data structure? [2 marks]

Answer

In-depth first traversal, the search in the graph starts with a root node and proceeds all the way down to the lowest level and then comes back up to visit the other nodes at a level. In other words, the traversal gives priority to the depth than the breadth of a node.

11. Consider the following graph among the following sequences:



List at least three sequences of depth-first traversals of the above graph. [2 marks]

Answer

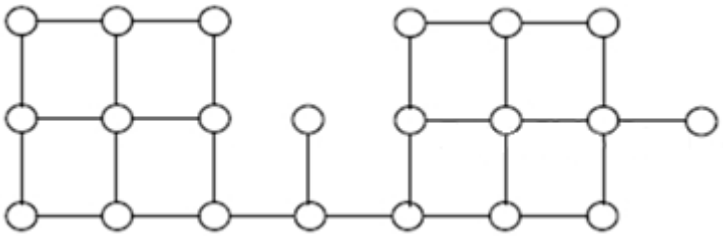
Some of the DFS sequences are:

i. a-b-e-g-h-f.

ii. a-b-f-h-g-e.

iii. a-f-g-h-b-e.

12. Suppose depth-first search is executed on the graph below starting at some unknown vertex. Assume that a recursive call to visit a vertex is made only after first checking that the vertex has not been visited earlier. Then the maximum possible recursion depth (including the initial call) is: _____. [2 marks]



Answer

Using DFS, the maximum possible recursion depth (including the initial call) is 19.

B. Graph Programming

1. Comment on the functionalities of the code segments given below. [5 marks]

Program:

```
public class SNPerson implements Iterable<SNPerson>{
private String name;
private Set<SNPerson> friends;

public SNPerson(String nm){
this.name = nm;
this.friends = new HashSet<SNPerson>();
}
public String getName() { return name; }
public void addFriend(SNPerson fr) {friends.add(fr); }
public void removeFriend(SNPerson fr) {friends.remove(fr); }
public boolean hasFriend(SNPerson fr) { return friends.contains(fr); }
public Iterator<SNPerson> iterator() { return friends.iterator(); }
```

Solution

The functionalities of the code segments given are:

- `private String name` and `private Set<SNPerson> friends` – property to store node and link.
- public String getName() { return name; }, public boolean hasFriend(SNPerson fr) { return friends.contains(fr); } - get() method for query purpose.
- public void addFriend(SNPerson fr) {friends.add(fr); }, public void removeFriend(SNPerson fr) {friends.remove(fr); } - set() method for forming link between nodes.

- this.friends = new HashSet<SNPerson>() - used in the object constructor to store the unique elements and it doesn't maintain any specific order of elements.
- public Iterator<SNPerson> Iterator() (+ Iterable<SNPerson>) – iteration method in an iterable object that allows you to traverse a collection of elements one-by-one.

2. Compare the following three programs that are used to establish relationship between persons in the social network graph (i.e. set up edges among the nodes in the graph). We need to write both lines of the code for bidirectional link relationships between persons in the social network. For each program, identify its characteristics. [3 marks]

Program 1:

```
public void addFriend(SNPerson fr) {
…
SNPerson P1 = new SNPerson("Jane");
SNPerson P2 = new SNPerson("Jay");
…
P1.addFriend(P2);
fr.addFriend(this);
}
```

Program 2:

```
public void addFriend(SNPerson fr) {
…
SNPerson P1 = new SNPerson("Jane");
SNPerson P2 = new SNPerson("Jay");
…
P1.addFriend(P2);
}

public void addFriendship(SNPerson fr){
this.friends.add(fr);
fr.friends.add(this);
}
```

Program 3:

```
public void addFriend(SNPerson fr) {friends.add(fr); }
…
SNPerson P1=new SNPerson("Jane");
```

```
SNPerson P2=new SNPerson("Jay");
…
P1.addFriendship(P2);
}

public void addFriendship(SNPerson fr){
this.addFriend(fr);
fr.addFriend(this);
}
```

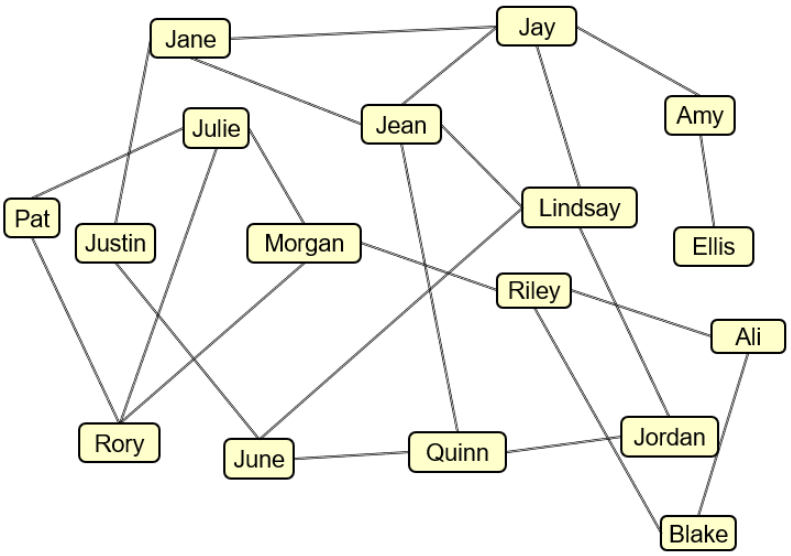
Solution

Program 1: The program calls for itself repeatedly as it is a recursive loop.

Program 2: The program still calls for other method than predetermined method.

Program 3: The link in the social network is established both ways from inside the predetermine method.

3. If the graph is as shown in the figure below, create the Java program that traverses its nodes and edges from node Jane. State the outcome of each program and comment on its characteristics.



a. Keep the visited nodes in an array. [10 marks]

b. Keep the visited flag inside node. [10 marks]

Solution

a. The program that traverses the graph that keeps the visited nodes in an array is as shown

below.

```
public void printNetwork(SNPerson person){
      printNetwork(person, new HashSet<SNPerson>());
}

public void printNetwork(SNPerson person, Set<SNPerson> visited){
visited.add(person);
      for (SNPerson friend : person){
            if (!visited.contains(friend) ){
                  printNetwork(friend, visited);
            }
      }
}
```

The program outcome is:

Jane, Jay, Amy, Ellis, Jean, Lindsay, Jordan, Quinn, June, and Justin.

Comment on the program:

- It shows all connected nodes in the connected graph.
- If we start from node Jane, it shows all links in the given graph connected to node Jane.
- What about nodes Pat, Julie, Rory, Morgan, Riley, Ali and Blake? It still doesn't work especially the graph is not connected!

b. The program that traverses the graph that keeps the visited flag inside the node is as shown below

```
public void printNetwork(SNPerson person){
      person.visit();
      for (SNPerson friend : person){
            if ( ! friend.isVisited()){
                  printNetwork(friend);
            }
      }
}
```

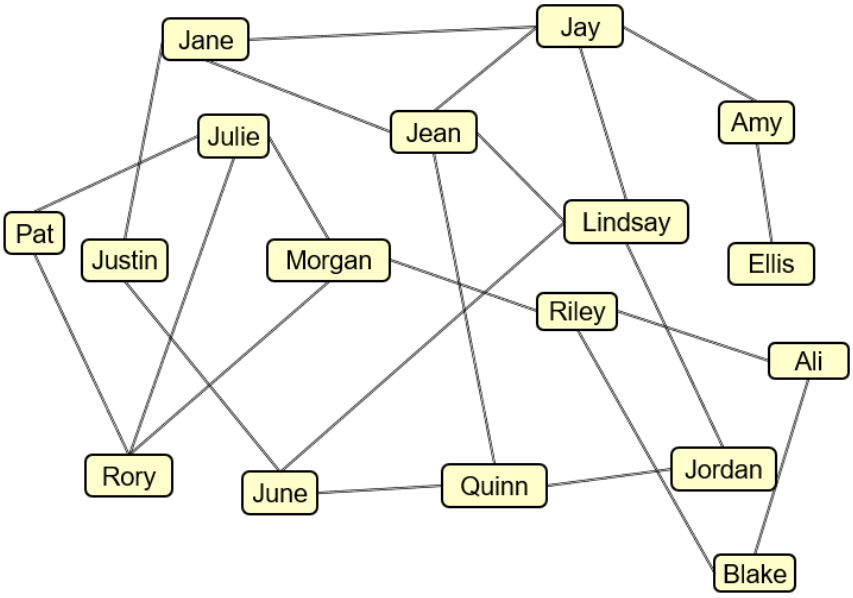
Program Outcome:

Jane, Jay, Amy, Ellis, Jean, Lindsay, Jordan, Quinn, June, and Justin.

Comment on the program:

- It shows all connected nodes in the connected graph.
- Need to calculate the total number of nodes.
- If you need to start traversing the graph all over again, we need to reset all the visited flags at the end!

4. If the social network graph is as shown in the figure below, design a program that will count the number of connected nodes. [6 marks]



Solution

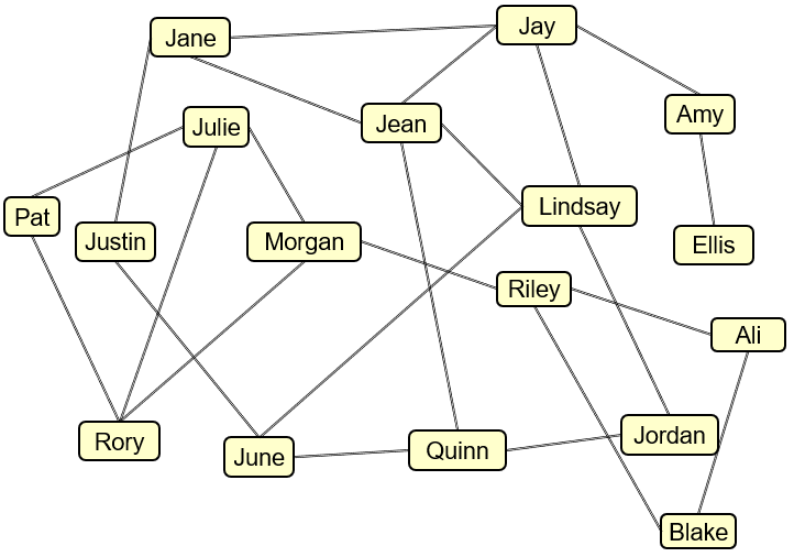
The program that will count the number of connected nodes is as shown below.

```
public int countConnected(SNPerson person){
   person.visit();
   int count =1;

   for (SNPerson friend : person){
         if ( ! friend.isVisited()) {
               count += countConnected(friend);
         }
```

```
    }
    return count;
}
```

5. Write a Java program for the social network graph that determine whether two people are connected in the network.



a. Recursive method. [10 marks]

b. Iterative method. [10 marks]

Solution

a. The recursive program is a shown below.

```
public boolean connectedTo(SNPerson person, SNPerson query){
      return connectedTo(person, query, new HashSet<SNPerson>());
}
public boolean connectedTo(SNPerson person, SNPerson query,
Set<SNPerson> visited){
      if (person.equals(query) ) {
            return true;
      }
      visited.add(person);

      for (SNPerson friend : person){
            if ( ! visited.contains(friend)  &&  connectedTo(friend,
```

```
        query, visited) ) {
                return true;
            }
        }
        return false;
    }
```

b. The iterative program is a shown below.

```
    public boolean connectedTo(SNPerson person, SNPerson query){
        Stack<SNPerson> stack = new ArrayDeque<SNPerson>();
        Set<SNPerson> visited = new HashSet<SNPerson>();

        stack.push(person);

        while (! stack.isEmpty()){
            SNPerson p = stack.pop();
            visited.add(p);

            if (p.equals(query) ) { return true; }

            for (SNPerson friend : p){
                if ( ! visited.contains(friend)) {
            stack.push(friend); }
            }
        }
        return false;
    }
```