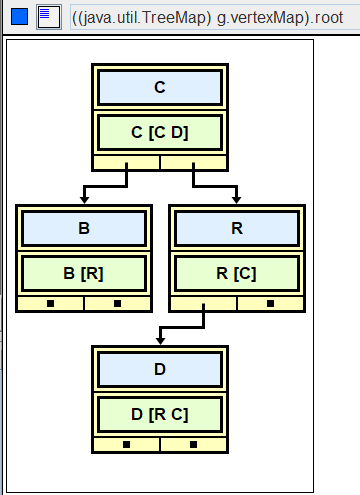
**Graphs 4:**

**Edge-List: DFS and BFS**

As we have mentioned before, a common task for graphs is to find the reachable vertices. This time we exhaustively traverse the edge-list representation, following the edges of each vertex until we reach an end or an already visited vertex.



|  |  |
| --- | --- |
| R | [C] |
| B | [R] |
| C | [C D] |
| D | [R C] |

What can you reach from D? \_\_\_\_\_\_\_\_\_

Traversals come in two varieties, a *depth-first* search and a *breadth-first* search.

**Depth-first search**

A *depth-first* search traverses a graph and returns a string containing the names of the reachable vertices. The order in which the vertices are processed is determined by a *stack*.

In what order are the vertices visited, if Vertex D is the source?

|  |  |
| --- | --- |
| B | [R] |
| C | [C D] |
| D | [R C] |
| R | [C] |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

|  |
| --- |
| first, create both a **list** of reachable vertices and a **stack.** |
| push D on the stack. |
| while the stack is not empty |
| pop a vertex. If the vertex isn't in the list, put it in the list.  warning: contains depends on hashCode and equals(Object) |
| process the vertex’s edges: push each vertex on the stack |
| concatenate the names in the list of reachable vertices.  return that string |

Write the code:

public String depthFirstSearch(String name)

**Breadth-first search**

A *breadth-first* search also traverses a graph and returns a string of the names of reachable vertices. The order in which the vertices are processed is determined by a *queue*. The algorithm is the same, but use a queue instead of a stack, changing the appropriate method calls. You should be able to visualize why a stack produces a depth-first search and a queue produces a breadth-first search. You should know that the breadth-first search produces the shortest path, shortest in the sense of visiting the fewest vertices (or edges.)

In what order are the vertices visited, if Vertex D is the source?

|  |  |
| --- | --- |
| B | [R] |
| C | [C D] |
| D | [R C] |
| R | [C] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

|  |
| --- |
| first, create both a **list** of reachable vertices and a **queue.** |
| enqueue D’s adjacent vertices. |
| while the queue is not empty |
| dequeue a vertex. If the vertex isn't in the list, put it in the list.  warning: contains depends on hashCode and equals(Object)on the name |
| process the vertex’s edges: enqueue each adjacent vertex. |
| concatenate the names in the list of reachable vertices.  return that string |

**Two Typical Questions on Graphs** Consider this graph:

v0 <------- v2

/ \

/ \

-> v1 <-/ \-> v4

/ \

/ \

/ \->v3 -------> v5

/ /

/ /

v6 <---------/

1. In what order are the vertices visited for a depth-first search (DFS) that starts at v0? (The neighboring edges are stored in alphabetic order.)
2. In what order are the vertices visited for a breadth-first search (BFS) that starts at v0? (The neighboring edges are stored in alphabetic order.)

**Assignment**

In AdjList, comment in the DFS\_BFS interface in the header and implement those two methods.

public class AdjList implements AdjListInterface, DFS\_BFS//,EdgeListWithCities

Here is the DFS\_BFS interface:

interface DFS\_BFS  
{  
 public String depthFirstSearch(String name);  
 public String breadthFirstSearch(String name);  
 /\* extra credit \*/  
 // public String depthFirstRecur(String name);  
 // public List<Vertex> depthFirstRecurHelper(Vertex v, List<Vertex> reachable);  
}

**Sample Run** (DFS\_BFS\_Driver.java using AdjList )

|  |
| --- |
| Edge List Representation!  B [R]  C [C D]  D [R C]  R [C]  Depth First Search Enter source: B  Reachables: B R C D  Enter source: C  Reachables: C D R  Enter source: D  Reachables: D C R  Enter source: R  Reachables: R C D  Enter source: -1   Breadth First Search Enter source: B  Reachables: B R C D  Enter source: C  Reachables: C D R  Enter source: D  Reachables: D R C  Enter source: R  Reachables: R C D  Enter source: -1  Set the debugger in AdjList to see the difference: D ---> D C R  D ---> D R C |

**Extension**

The iterative DFS algorithm above used a stack. Recursion also behaves like a stack, processing each method call in reverse. You have used this kind of recursion in previous labs: for every vertex you visit, process it, then recur (in reverse) on each of its edges. Return the string of the names of the reachable vertices.