Project 5: Graph algorithms (*continued*)

Deadlines: submit your files **electronically** by **midnight** (end of day) on **Wednesday, 06/02/21**

Late submission: you can submit your work within 24 hours after deadline (penalty applies):
-by 2 a.m. on 06/03/21 for **6-point penalty**,
-from 2:01 a.m. till *noon* on 06/03/21 for **15-point penalty**,
-from 12:01 p.m. till *midnight* (end of the day) on 06/03/21 for **30-point penalty**

Programming Language: ***Java***.

Part 3 of 4 (30 points): implementation of breadth-first-search and related routines

1. Make additions in the *DiGraph* class to include the implementation of the BFS algorithm and related routines to allow determining shortest distances and paths between vertices of the graph.

Note: For the purpose of **this** assignment, we'll implement the *BFS* algorithm as a ***private*** method.

Add the following content to the *DiGraph* class's definition. Note that parameter vertices in ***c,d,e*** methods are given via **natural** numbering: you need to manage *adjustments* with Java indexing (no validity check).

a) A ***private*** nested class ***VertexInfo*** with two *int* instance variables: one for the distance (or length of the path), and the second for the predecessor/parent of a vertex.

b) A ***private*** method ***BFS(int s)***: returns an array of *VertexInfo* type objects containing data that can be used to construct shortest paths from *s* vertex to all vertices in the graph that are reachable from s. This is the BFS algorithm we discussed in class (see lecture handout).
Note: this is a private method so *you* can decide how *s* vertex is given (via natural numbering or not).

Attention: In *BFS* you need a regular queue (**not** a priority queue). To implement a queue, in Java you can define an object of ***LinkedList*** class (the list is for integers). Your list will function like a regular queue if you always add an element to the end of the list (*addLast* method) and delete an element from the front of the list (*removeFirst* method).

c) A *public* method ***isTherePath(int from, int to)***: returns *true* if there is a path from *from* vertex to *to* vertex, and *false* otherwise.
Hint: invokes *BFS* method and uses data in the returned array.

d) A *public* method ***lengthOfPath(int from, int to)***: returns an integer – the shortest distance of the *to* vertex from the *from* vertex.
Hint: invokes *BFS* method and uses data in the returned array.

e) A *public* method ***printPath****(int from, int to)*: arranges the output of the shortest path from *from* vertex to *to* vertex if *to* is reachable from *from* (vertices of the path should be printed in **natural** numbering); it outputs "There is no path" otherwise.
Hint: invokes *BFS* method and uses data in the returned array.

2. Make additions to the *DiGraphTest* class's *main* method to incorporate three additional services.
Add three menu choices and arrange the execution of the following three **new** services:

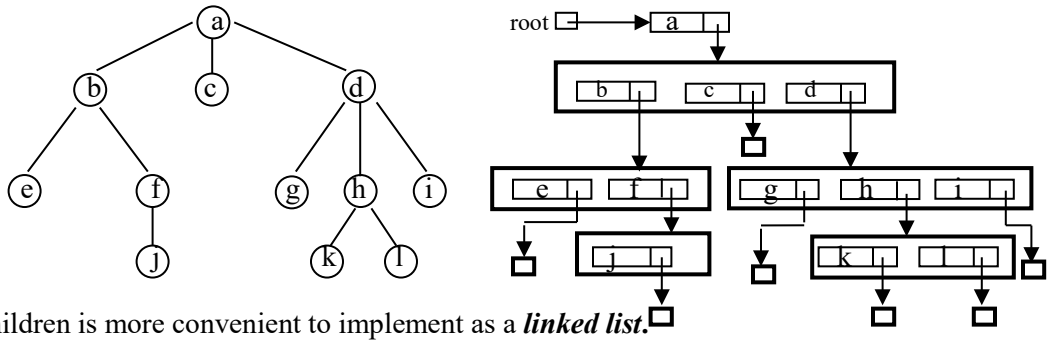
- For given two vertices find out if there is a path going from the first vertex to the second.
- For given two vertices find the length of the shortest path going from the first vertex to the second.
- For given two vertices print the shortest path going from the first vertex to the second, if such path exists; and output an informative message otherwise.

Attention: 1. You must use menu letters: ***i*** - **i**s there a path, ***l*** - **l**ength of the path, and ***s*** - **s**hortest path.
2. To execute *each* of these requests, you need to ask the user to enter 2 vertex numbers.
Note: vertices are given via **natural** numbering, i.e. **starting with 1**. No validity check.
3. For *l* menu choice, if the destination vertex is not reachable from the source vertex, you can output -1 as the length of the path.

3. Edit, compile and execute *DiGraphTest* class, testing all new services very thoroughly.

Part 4 of 4 (30 points): building and printing of the breadth-first-tree

Preface:
A general tree can be implemented as a linked structure of nodes where each node contains a **value** and a **link to a *list* of its children**. Here is an illustration of the linked structure for the given graph:



Notes: 1. List of children is more convenient to implement as a ***linked list.***
2. In this example the list of children for *c,e,g,i,j,k,l* nodes is empty.

Assignment:

1. Make additions in the *DiGraph* class to include the implementation of routines that will allow to print the *breadth-first-tree* of the graph for the given start vertex s.

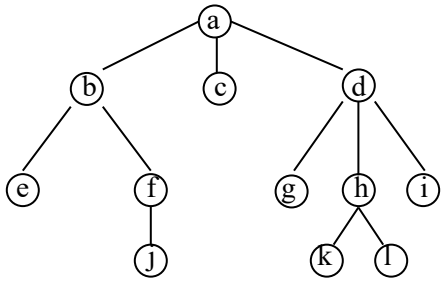
Add the following members to the *DiGraph* class's definition.

a) A ***private*** nested class ***TreeNode*** with two instance variables: an *int* type variable to hold the vertex-number and a *LinkedList* type list to hold *TreeNode* type objects representing this vertex's children.

b) A ***private*** method ***buildTree(int s)***: returns the **root** of the breadth-first-tree for the given *s* source-vertex. The tree can be built based on the data in the array returned by the BFS method.
Note: this is a private method so *you* can decide how *s* vertex is given (via natural numbering or not).
Hint: Create *TreeNode* type objects for every graph-vertex and link them using the array returned by the BFS method, namely, using the *predecessor* values in this array (all you need to do is go through the array and make arrangement to include each vertex into its parent's list of children).

c) A *public* method ***printTree(int s)***: this method prints the breadth-first-tree for a given source vertex s. Vertex *s* is given via **natural** numbering: manage *adjustments* with Java indexing. No validity check.
Hint: invoke *buildTree* method and obtain the breadth-first-tree (more precisely, its root-node). Then arrange the printing of this tree in the required format (vertices must be **naturally** numbered)

Format of the tree: the tree should be printed in this commonly used formats (see the picture below) where levels are depicted vertically (with 0-level as the leftmost "column"); elements on each level are depicted in a "column" in the same order (with the leftmost element of the level depicted as the highest one). You are **required to implement the proper indentation**, i.e. each level of the tree, compared to the previous level, needs to be indented to the right by few spaces (e.g. 4 spaces).




```
a
    b
        e
        f
            j
    c
    d
        g
        h
            k
            l
        i
```

Attention: Printing of the tree is *easily* implemented with **recursion**. You can have this *public* method obtain the tree and initiate the recursion, and then *include* a private method to carry the recursive work.

2. **Make additions to the *DiGraphTest* class's *main* method to incorporate one new service.**
   Add one new menu choices and arrange the execution of the following **new** service:
   - for the given start vertex print the breadth-first-tree.

   Attention: 1. Must use menu letter ***b*** for this new option: print **b**readth-first-tree.
   2. To execute this request, you need to ask the user to enter a source vertex number (**natural** number). There is no need for validity check.

3. **Edit, compile and execute *DiGraphTest* class testing this new service very thoroughly.**

Submitting your work:

Turn in *DiGraph* and *DiGraphTest* source files electronically via "*handin*" procedure by the deadline (see the top of the first page of this document for the due date and time):

The account *id* is: ***hg-cpe103***
The *name* of the assignment is: ***Project5***
The *name* of the assignment for late submissions is: ***Project5_late.***

Important requirements:

1. Your programs must **compile and run from command line** on our departmental servers. So, before submitting, make sure your programs compile and run on lab machines.
2. Do not create packages for your source code, do not zip, and do not use folders for your submission – all these complicate the automated grading process.
3. Each file must have a **comment-header** which should include both authors' **names** and **ids** (as in *id@calpoly.edu*), as well as the **date** and the **assignment name** (e.g. Project 5). **Reminder**: only **one** submission needs to be made for a pair/team.
4. The header of each file must **start at the first line** (no empty lines or *import* statements before it) and should be followed by at least one empty line before the first line of the code.