COSC1285/2123: Algorithms & Analysis
Laboratory 4

Topic

Graph searching, BFS and DFS.

Objective

Students who complete this lab should:

- Learn to implement depth first search (DFS) and breadth first search (BFS) traversal.
- Learn to use breadth first search (BFS) for computing shortest path distances in graphs.

Introduction

In this lab exercise you will complete the implementation for DFS and BFS traversals, and use BFS to compute the shortest path between to vertices in a graph.

DFS

Depth first search can be implemented as recursive algorithm, as outlined in Algorithm 1.

Algorithm 1 Depth first search algorithm.

ALGORITHM **DFS** (G)
Perform a Depth first search traversal of a graph.

INPUT : Graph $G = \langle V, E \rangle$, seed/starting vertex s.
OUTPUT : Graph G with its vertices marked with consecutive integers in the order they were visited/processed.

// *variable that stores the visitation order*
$order = 0$
// *mark all vertices unvisited*
for $i = 0$ **to** v **do**
    $Marked[i] = 0$
end for
DFSR (s)

ALGORITHM **DFSR** (v)
Recursively visit all connected vertices.

INPUT : A seed/starting vertex v
OUTPUT : Graph G with its vertices marked with consecutive integers in the order they were visited/processed.

```
Marked[v] = order
order = order + 1
for w ∈ V adjacent to v do
    if not Marked[w] then
        DFSR (w)
    end if
end for
```

BFS

Breadth first search can be implemented using a queue, which computes the correct order that the vertices should be visited or processed. Algorithm 2 outlines the approach.

Algorithm 2 Breadth first search algorithm.

ALGORITHM **BFS** (G)
Implement a Breadth First Traversal of a graph.

INPUT : Graph $G = \langle V, E \rangle$, seed/starting vertex s.
OUTPUT : Graph G with its vertices marked with consecutive integers in the order that they are traversed/visited.

```
// variable that stores the visitation order
order = 0
// mark all vertices unvisited
for i = 0 to v do
    Marked[i] = 0
end for
Queue.enqueue(s)
while not Stack.isEmpty() do
    v = Queue.deque()
    Marked[v] = order
    order = order + 1
    for w ∈ V adjacent to v do
        if not Marked[w] then
            Queue.enqueue(w)
        end if
    end for
end while
```

Shortest path distance

The shortest path between two vertices in an unweighted graph is the path that traverses the least number of edges to go from a *source* vertex to a *target* vertex. The shortest path distance between two vertices is the number of edges traversed in an unweighted graph.

Computing shortest path distances using BFS

To compute shortest path distances using BFS, a minor modification is needed to the previous pseudo code for BFS. We know that the neighbours of seed (source) vertex s has distance 0 from s (itself). Its neighbours have a shortest path distance of 1, their neighbours have a shortest path distance of 2 from the source vertex and so on.

To implement this, when we enqueue the vertex v, we also need to store the shortest path distance from s to v, so that when we deque it, then we can just increment the shortest path distance by 1 for its neighbours.

Shortest path distance using BFS can be implemented as outlined in Algorithm 3.

Algorithm 3 Computing shortest path distance using Breadth first search.

ALGORITHM **ShortestPathDistance** (G,s,t)
Compute the shortest path distance between source vertex s to target vertex t using BFS.

INPUT : Graph $G = \langle V,E \rangle$, source vertex s, target vertex t. starting OUTPUT : Shortest path distance between vertices s and t. If s and t are disconnected, than return -1.

```
// mark all vertices unvisited
for i = 0 to v do
    Marked[i] = 0
end for
// initial distance from source vertex is 0
Queue.enqueue((s, 0))
while not Stack.isEmpty() do
    (v, d) = Queue.deque()
    if v == t then
        return d
    end if
    Marked[v] = order
    for w ∈ V adjacent to v do
        if not Marked[w] then
            Queue.enqueue((w, d + 1)
        end if
    end for
end while
// If reach here, then either s and t are disconnected or t does not exist
return −1
```

Provided Code

The programs BFS, DFS and BFSSHORTESTPATH reads a sequence of white-space separated integers from file as edges, construct the corresponding graph. BFS runs breadth first search traversal on the input graph, DFS runs depth first search traversal and BFSSHORTESTPATH computes the shortest path distance between two input vertices.

file	description
`Graph.java`	Code to implement a graph, using adjacency list representation. No need to modify this file.
`BFS.java`	Code implementing breadth-first-traversal. You are to **implement the BFS traversal functionality**, using Algorithm 2.
`DFS.java`	Almost complete code implementing DFS traversal. You are to **implement the DFS traversal functionality**, using Algorithm 1.
`BFSShortestPath`	Skeleton code for you to **implement the shortest path distance algorithm**, using Algorithm 3.

Compile the program using the following command:

```
javac *.java
```

To run BFS:

```
java BFS <graph fileName> <seed vertex>
```

To run DFS:

```
java DFS <graph fileName> <seed vertex>
```

To run BFSShortestPath:

```
java BFSShortestPath <graph fileName> <source vertex> <target vertex>
```

We have provided a sample graph file for you to experiment with. It is called `graph1.in`. It contains vertices 0 to 9. Using this file, as an example, you can run BFS on this graph on seed vertex 1 as follows:

```
java BFS graph1.in 1
```

Task

Your task in this lab exercise is to implement the following algorithms in `DFS.java`, `BFS.java` and `BFS-ShortestPath`:

- DFS.java: Implement `traverse(Graph g, int s)`, to compute a depth first traversal from seed vertex "s".
- BFS.java: Implement `traverse(Graph g, int s)`, to compute a breadth first traversal from seed vertex "s".
- BFSShortestPaths.java: Implement `spd(Graph g, int s, int t)` to compute the shortest path distance between vertices "s" and "t".