# Lab 13 – Data Structures and Algorithms

## Group 01:

### Shahzaib Irfan, 2021 – CS – 07

### Afraz Butt, 2021 – CS - 12

### Muhammad Hamza, 2021 – CS – 41

# Part 3 – Badger Badger Badger

## (a):

PathFind(G , s , t , x):

bool flag = false

Initialize a queue Q with S as its first element.

While Q != NIL:

element = Q.dequeue()

For each neighbour in Adj[element]:

If (x <= tunnelWidth(element, neighbour)):

Q.enqueue(neighbour)

Flag = true

Break

Else if neighbour is t: break

Else:

Flag = false //Badger can not be accommodated through here

if flag is false and Q is empty:

return false // after a point there was no width for passing

return true

## (b):

PathFind-LargestRealNumber(G , s , t , x)

Initialize minWidthReq = 0

Initialize a queue Q with s as its first element

While Q != NIL:

element = Q.dequeue()

for each neighbour in adj[element]:

if neighbour is T: break

if(x <= tunnelWidth(element, neighbour)):

if (minWidthReq + tunnelWidth(element, neighbour) <= x and minWidthReq <= tunnelWidth(element , neighbour)):

minWidthReq += tunnelWidth(element , neighbour)

/\*This step adds minimum Widths up till that point\*/

Return minWidthReq

## Explanation of Algorithms:

## (a):

The subroutine involves checking each width of neighbors. If a badger can be accommodated in the width of tunnel, we keep the flag is true else it stays false. At the end if it is false, no path was found and we return false, else we return true.

## (b):

Now the subroutine is slightly modified. Instead of seeking a pure path, we now need to find a largest Real Number such that it defines badger width and tunnel width. This is easy, subject to a constraint that minWidthReq variable upon adding to the widths, must not decrease badgerWidth.

# Part 2 – Topological Sorting

## a):

### a):

CDEAB

### b):

ABCDE

## b):

### a):

CEDBA

### b):

ABCDE