NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC1007 Data Structures and Algorithms **2021/22 Semester 2**

Solution 5: BFS & DFS

School of Computer Science and Engineering *Nanyang Technological University*

Q1 Manually execute breadth-first search on the undirected graph in Figure 5.1, starting from vertex s. Then, use it as an example to illustrate the following properties:

(a) The results of breadth-first search may depend on the order in which the neighbours of a given vertex are visited.

(b) With different orders of visiting the neighbours, although the BFS tree may be different, the distance from starting vertex s to each vertex will be the same.

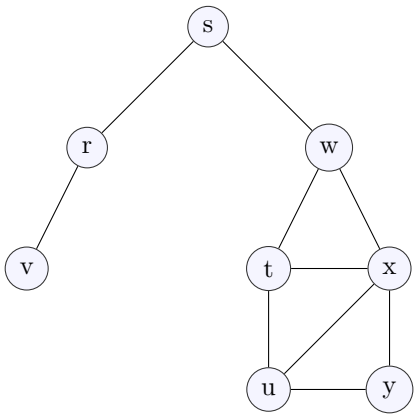

Figure 5.1: The graph for Q1

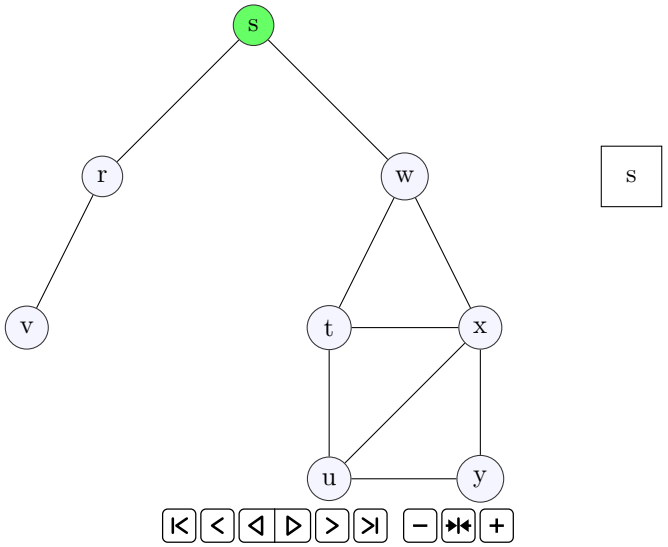

Figure 5.2: Graph for S1

S1

- When the queue is empty, the BFS is finished.
- The edges of BFS tree are shown in red.
- Likewise, a BFS tree can be constructed if the neighbors are visited in the reverse alphabetical order (an exercise for the students).
- The two trees differ in that vertex u is adjacent with t in the left tree, but adjacent with x in the right tree.
- The distance from starting vertex s to each other vertex is equal in the two trees.

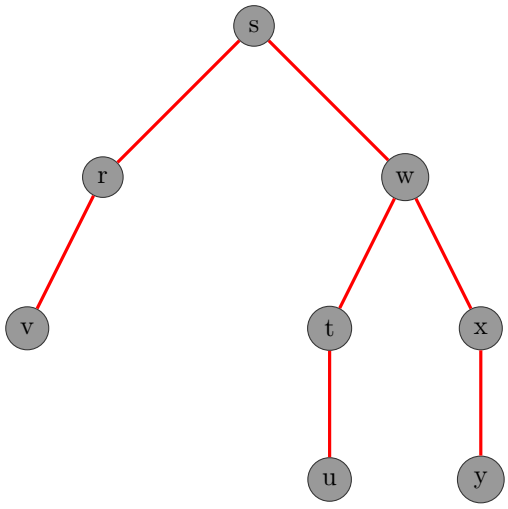

Figure 5.3: Visiting neighbors in alphabetical order

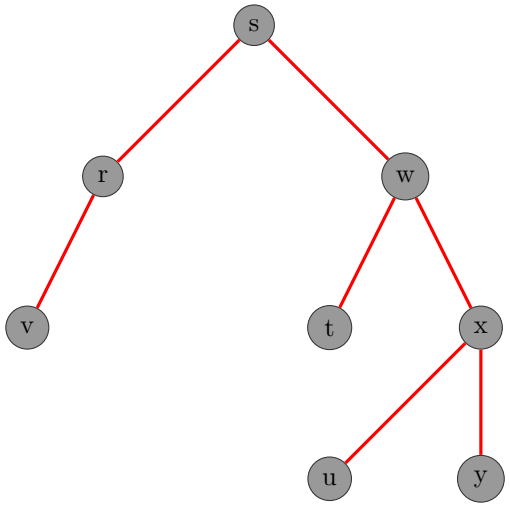

Figure 5.4: Visiting neighbors in reverse alphabetical order

Q2 Give a pseudocode of checking whether an undirected graph is connected or not by using breadth-first search.

S2 Idea: Apply BFS in any vertex to find the spanning tree, and check whether all the vertices of the graph are in this tree.

Algorithm 1 Check connected-graph (BFS)

```
function ConnectGraph(Graph G)
    create a Queue, Q
    select an arbitrary vertex v
    enqueue v into Q
    mark v as visited
    while Q is not empty do
        dequeue a vertex from Q and denoted as w
        for each unvisited vertex u adjacent to w do
            mark u as visited
            enqueue u into Q
        end for
    end while
    for each vertex z in G do
        if z is not visited then
            return False
        end if
    end for
    return True
end function
```

Q3 Give a pseudocode of finding a simple path connecting two given vertices in an undirected graph by using depth-first search.

S3 Idea: Use DFS to traverse the paths starting from v and check whether any of them contains w.

Algorithm 2 Simple path (DFS)

```
function SimplePath(Graph G, Vertex v,Vertex w)
    create a Stack, S
    push v into S
    mark v as visited
    while S is not empty do
        peek the stack and denote the vertex as x
        if x == w then
            while S is not empty do
                pop a vertex from S
                peek the stack
                print the link
            end while
            return Found
        end if
        if no unvisited vertices are adjacent to x then
            pop a vertex from S
        else
            push an unvisited vertex u adjacent to x
            mark u as visited
        end if
    end while
return Not Found
end function
```