Experiment 2

Path traversal using Breadth First Search

Foundations of AI

By- Shivansh Jain, VIT Chennai

AIM:-

To find the path traversed using BFS (Breadth First Search) given the graph and the starting node

CONCEPT:-

Breadth-first search is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root(any selected node in case of a graph) and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

ALGORITHM:-

1. Created a queue of nodes and visited array.
2. Insert the root in the queue. Run a loop till the queue is not empty.
3. Pop the element from the queue and append in path.
4. For every child and unvisited node of current node, mark the node and insert it in the queue.

STEP BY STEP IMPLEMENTATION:-

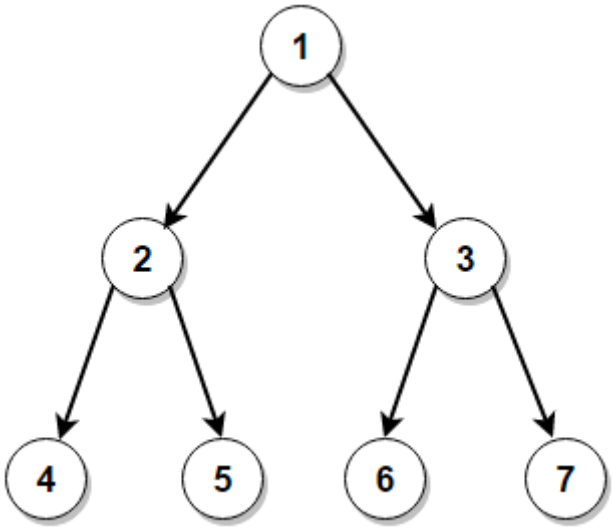
```
#R version 4.1.0
#RStudio version 1.4.1717

rm(list = ls())
#to ensure a clean environment before executing the code

bfs <- function(graph, start){
  # function dfs with arguments graph and start
  # graph is an adjacency-matrix-representation of the graph where (x,y) is TRUE if there is an edge
between nodes x and y
  # start the node to start from.
  # returns an array containing te path from the given start node till it traverses every node in the graph


  #using a queue to manage the nodes that have yet to be visited, intialized with the start node
  queue = c(start)

  #array path to store the path
  path=c()

  # A boolean array indicating whether we have already visited a node
  # the start node is already visited
  visited = rep(FALSE, nrow(graph))
  visited[start] = TRUE
```

```
  # while there are nodes yet to visit
  while(length(queue) > 0) {
    #get node to explore
    #remove the node from queue
    node = queue[1]
    queue = queue[-1]
    # the node is added to the path
    # then we check all the neighbouring elements of the node which are yet to be visited and add them to
the stack
    path = c(path,node)
    for(i in seq_along(graph[node,])) {
      if(graph[node,i] && !visited[i]){
        visited[i] = TRUE
        queue = c(queue, i)
      }
    }
  }
  # return path
  return (path)

}


#driver code

test_case_1 <-
matrix(c(FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,
FALSE,FALSE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,F
ALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,F
ALSE,FALSE), nrow = 7,byrow = TRUE)
test_case_2 <-
matrix(c(FALSE,TRUE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,TRUE,FALSE,T
RUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE,TRUE,TRUE,TRUE,FALSE,TRUE,FALSE,FALSE,FALSE,FALSE
,FALSE,TRUE,FALSE,FALSE), nrow = 6,byrow = TRUE)
path_1=bfs(test_case_1,1)
path_2=bfs(test_case_2,6)
cat("Final path(test_case_1): ", path_1)
cat("Final path(test_case_2): ", path_2)
```

RESULTS AND OUTPUT:-

→

Test case 1:-



Graph:-

Filter

	V1	V2	V3	V4	V5	V6	V7
1	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
2	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
3	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
4	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
5	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
6	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
7	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

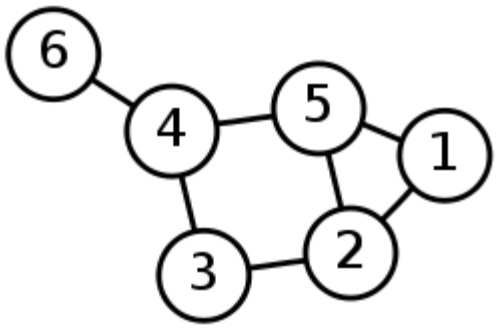
Output :-

Starting node:- 1

```
> path_1=bfs(test_case_1,1)
> cat("Final path(test_case_1): ", path_1)
Final path(test_case_1):  1 2 3 4 5 6 7
>
```

→

Test case 2:-



Graph

Filter

	V1	V2	V3	V4	V5	V6
1	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
2	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
3	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
4	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE
5	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
6	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE

OUTPUT

Starting node:- 6

```
> path_2=bfs(test_case_2,6)
> cat("Final path(test_case_2): ", path_2)
Final path(test_case_2):  6 4 3 5 2 1
>
```