Traversing a Graph

⇒ By traversing a graph means the method of examining the nodes and edges of the graph.

⇒ There are 2 standard methods of a graph traversal —

(1) Breadth-first search (BFS)

(2) Depth-first search (DFS)

During the execution of algorithms, each node N of G will be in one of the three states, called the status of N —

Status = 1 : (Ready State) : The initial state of the node N

Status = 2 : (waiting state) : The node N is on the queue or stack waiting to be processed

Status = 3 : (processed state) : The node N has been processed.

BFS (Breadth-first search)

⇒ This search algorithm begin at the root-node and explores all the neighbouring nodes.

⇒ For example if we start from root node 'A' then we examine all the neighbours of A then we examine all the neighbours of neighbours of A and so on, until it finds the goal.

⇒ This means that we need to track of the neighbours of a node and guarantee that every

node in the graph is processed and no node is processed more than once.

* This is accomplished by using Queue to hold the nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node.

Algorithm:

(1) Initialise all nodes to the ready state (STATUS = 1)

(2) Put the starting node A in queue and change its status to the waiting state (STATUS = 2)

(3) Repeat steps (4) and (5) until queue is empty:

(4) Remove the front node N of queue. Process N and change the status of N to the processed state. (STATUS = 3)

(5) Add to the rear of queue, all the neighbours of N that are in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2)

[End of the step (3) loop]

(6) Exit.

FRONT = 7 QUEUE: A, F, C, B, D, G, E
REAR = 7 ORIG: Ø, A, A, A, F, B, G

step (8) Remove the front element E from QUEUE and add to QUEUE the neighbours of E

FRONT = 8 QUEUE: A, F, C, B, D, G, E, J
REAR = 8 ORIG: Ø, A, A, A, F, B, G, E

* We stop as soon as J is added to QUEUE, since J is our final destination
* We now backtrack from J using the ORIG array to find the path P.

Thus - J ← E ← G ← B ← A is the required path.

Depth-first search: (DFS)

* This algorithm begins with starting node A
⇒ First examine the starting node A. then process a neighbour of A, then a neighbour of a neighbour of A and so on.

⇒ After coming to the end of the path P, backtrack on P until you can continue along another path P' and so on.

⇒ This algorithm uses a STACK to hold the nodes. STATUS is used to note current

Note: The DFS progresses by expanding the starting node of G and thus going deeper and deeper until a goal node is found, or until it reaches a node that has no children. When a dead end is reached, the alg. backtracks returning to the most recent node that has not been completely explored.

Algorithm: executes DFS on a graph G beginning at a starting node A

(1) Initialize all nodes to the ready state (STATUS=1)

(2) Push the starting node A onto STACK and change it's status to the waiting state (STATUS=2).

(3) Repeat steps (4) and (5) until STACK is empty.

(4) Pop the top node N of STACK. process N and change it's status to the processed state (STATUS=3)

(5) Push on to stack all the neighbours of N that are still in the ready state (STATUS=1) and change their status to the waiting state (STATUS=2)

[End of step 3 loop]

(6) Exit.

Example: take the same dig.

Suppose we want to find and print all the nodes reachable from the node J.
So our starting node = J

Step 1 Initially push J onto the stack.

Stack: J

Step 2 Pop and print top element J, then push on to the stack all the neighbours of J (those that are in ready state)

Print J Stack: D, K

Step 3 Pop and print the top element K, then push on to the stack all the neighbours of K

Print K Stack: D, E, G

Step 4 Pop and print the top element G, then push on to the stack neighbours of G.

Print G Stack: D, E, C

Note: only 'C' is pushed onto the stack, since the other neighbour, E is not in the ready state [E has already been put into the stack]

Step 5 Pop and print top element C, then push onto the stack the neighbours of C

Print C Stack: D, E, F

Step 6 Pop and print the top element F

Print F Stack: D, E

Note D' is already pushed so it is not considered.

Step (7) pop and print the top element E.
Note: (None of the 3 neighbours of E is in the ready state)

Print E Stack: D

Step (8) pop and print the top element D.
Push on to the stack, all the neighbours of D.
Print D Stack:

* The stack is now empty so the DFS of G starting at J is now complete.

Accordingly, the nodes which are printed,

J, K, G, C, F, E, D

are precisely the nodes which are reachable from J