Python - Graph Algorithms

Graphs are very useful data structures in solving many important mathematical challenges. For example computer network topology or analysing molecular structures of chemical compounds. They are also used in city traffic or route planning and even in human languages and their grammar. All these applications have a common challenge of traversing the graph using their edges and ensuring that all nodes of the graphs are visited. There are two common established methods to do this traversal which is described below.

Depth First Traversal

Also called depth first search (DFS),this algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration. We implement DFS for a graph in python using the set data types as they provide the required functionalities to keep track of visited and unvisited nodes.

Example

```
class graph:
   def __init__(self,gdict=None):
      if gdict is None:
         gdict = {}
      self.gdict = gdict
# Check for the visisted and unvisited nodes
def dfs(graph, start, visited = None):
   if visited is None:
      visited = set()
   visited.add(start)
   print(start)
   for next in graph[start] - visited:
      dfs(graph, next, visited)
   return visited

gdict = {
   "a" : set(["b","c"]),
   "b" : set(["a", "d"]),
   "c" : set(["a", "d"]),
   "d" : set(["e"]),
   "e" : set(["a"])
}
dfs(gdict, 'a')
```

Output

When the above code is executed, it produces the following result −

```
a
b
d
e
c
```

Breadth First Traversal

Also called breadth first search (BFS),this algorithm traverses a graph breadth ward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration. Please visit this link in our website to understand the details of BFS steps for a graph.

We implement BFS for a graph in python using queue data structure discussed earlier. When we keep visiting the adjacent unvisited nodes and keep adding it to the queue. Then we start dequeue only the node which is left with no unvisited nodes. We stop the program when there is no next adjacent node to be visited.

Example

```
import collections
class graph:
   def __init__(self,gdict=None):
      if gdict is None:
         gdict = {}
      self.gdict = gdict
def bfs(graph, startnode):
# Track the visited and unvisited nodes using queue
   seen, queue = set([startnode]), collections.deque([startnode])
   while queue:
      vertex = queue.popleft()
      marked(vertex)
      for node in graph[vertex]:
         if node not in seen:
            seen.add(node)
            queue.append(node)

def marked(n):
   print(n)

# The graph dictionary
gdict = {
   "a" : set(["b","c"]),
   "b" : set(["a", "d"]),
   "c" : set(["a", "d"]),
   "d" : set(["e"]),
   "e" : set(["a"])
}
bfs(gdict, "a")
```

Output

When the above code is executed, it produces the following result −

```
a
c
b
d
e
```