BFS (adj list)

(4) visited = [0] * nodes, queue = [] $\rightarrow$ c

BFS (visited, graph, start, end):

Do visited [start] ← 1 $\rightarrow$ c

Do queue ← append (node) $\rightarrow$ c

while queue not empty: $\rightarrow$ goes for every vertex in graph.

part 1.

Do queue ← pop() $\rightarrow$ c

Print m. $\rightarrow$ c

if m = = end: $\rightarrow$ c

Break

else: $\rightarrow$ visits all edge of a vertex v.

visited[neighbor.-1] = 0 $\rightarrow$ c

Do visite [neighbour - 1] ← 1 $\rightarrow$ c

part 2

Do queue ← append (neighbour) $\rightarrow$ c.

so, time is, c + edges of V_i

per part 2, ~~in each round~~

where, i is |V|,

so, $\underbrace{(c + E_1) + (c + E_2) + (c + E_3) \ldots + (c + E_V)}_{|V|}$

$= c|V| + \Sigma E$

$= |V| + |E|$ as ΣE is $|E|$

so, $O(|V| + |E|)$

BFS (adj matrix)

for matrix, we form a n x n matrix and n is no. of vertices (v).

For traversal we iterate through each block to block and there are n^2 or V^2 blocks to traverse in total. So time complexity is

$O(V^2)$

DFS (adj list)

visited = [0] * nodes
printed = [ ]

$\Big\} \; c$

DFS - VISIT (graph, node)

    Do visited (node - 1) ← 1
    printed.append (node)

    $\Big\} \; c$

    for each node in graph(node) → runs for each vertex V.

        if node not visited

            DFS_VISIT(graph, node) → keeps visiting neighbouring nodes till none left, in-indepth manner.

def dfs(graph, end) :

    for node in graph

        if node not visited.

            DFS - visit (graph, node).

        print list till end. → $O(V)$

Time taken, $\underbrace{(c + E_1) + (c + E_2) + \dots (c + E_v)}_{V \text{ times.}}$

$= cV + \Sigma E$

$= O(V + E)$

DFS (adj matrix)

for matrix, same as BFS, we must traverse through all cells of $V \times V$ i.e V^2 cells in total. Thus, for adj matrix complexity is $O(V^2)$.

My rival Gary reaches victory road first, as,
We know DFS works well if the searched element
is far from the start vertex even though BFS and
DFS have the same time complexity. we can also
see this our outputs. where for DFS we
traverse less edges than we do for BFS as the
end destination is far from our start node.
for BFS we traverse 8 edges/vertices, for
DFS we traverse 6 vertices.

1. Yes, if I maintain a dictionary with the name and the corresponding place number, then I can just check sequence of the numbers and exchange them with the value of my new dictionary. that has the numbers a its' key.

eg. seq is

1	2	3	6	9	10

store_dict = { 1: dhaka, 2: china, 3: USA, 6:

2. We can run a loop to check unvisited vertex and the we will visit them using the normal BFS algorithm.

3. Using the blow condition we can detect cycle using DFS. Here, white mean unvisited vertex, grey means visited but not explored and black mean visited and explored. so

If (color (x) = 'Gray' and y not parent of x)
then cycle exists.