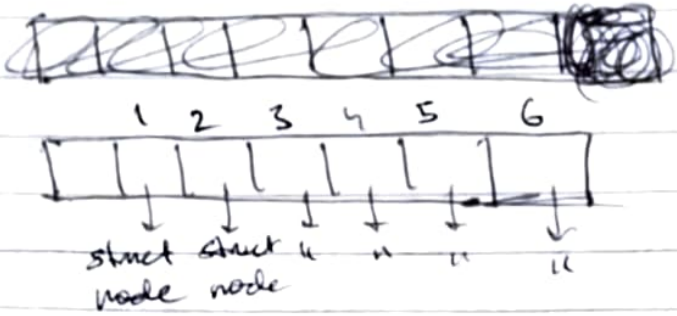
Aniket Gupta
2022073

HW - 10

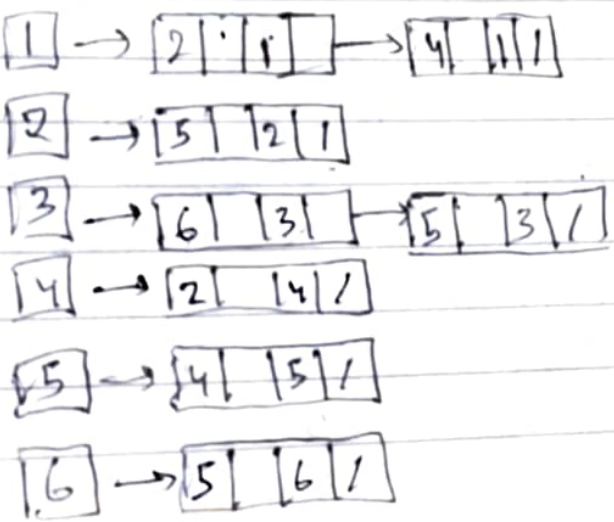
In order to store incoming edges of the adjacency list efficiently (i.e, without duplicating the payload), one very simple solution would be to make another field in our struct node that would contain the link to the array index pointing to our adjacency linked list for outgoing edges. We can simply store the array index in this case.

For the given example, suppose our array is :




```
struct node {
    int vertex;
    char payload[1024];
    int array_reference;
    struct node * next;
}
```

Hence, each node of the linked list contains incoming edges and this approach is very efficient since we store only an extra integer value



vertex 'u' such that its shortest distance u.d from the source vertex is equal to 'l' and the vertex color is 'BLACK'.

If such a vertex is found, it means that a non-circular path of length 'l' exists from the source vertex 's' to another vertex and our algorithm would return 'true', else, we return 'false'.

```
findPath(G, s, L)
// G is a graph (V, E)
// s is the source vertex
// L is the length of the path

    BFS(G, s)  // compute shortest paths from s

    if L == 0
        return true

    for each vertex u in G.V
        if u.d == L and u.color == BLACK
            return true

    return false
```

Q3. If an adjacency matrix were to be used to store the edges in the BFS algorithm :

- setting the colour, distance and predecessor fields would take $O(V)$ time
- Enqueueing and dequeueing vertices from the queue would take $O(1)$ time
- when iterating over the adjacency matrix, for each vertex, we would have to iterate over the entire row of the matrix. Since we have V vertices, time complexity would be $O(V \times V) = O(V^2)$

Hence, we have a total time complexity of $O(V^2)$

Q = ϕ or Q = [ ]

Q = [(1, 0)]

Q = [ ] u = 1, u.d = 0

Q = [(2, 1)] v = 2, v.d = 1 2 is black

Q = [(2, 1), (4, 1)] v = 4, v.d = 1 4 is black

Q = [(2, 1), (4, 1), (7, 1)] v = 7, v.d = 1 7 is black

1 is black

Q = [(4, 1), (7, 1)] u = 2, u.d = 1

Q = [(4, 1), (7, 1), (3, 2)] v = 3, v.d = 2 3 is black

Q = [(7, 1), (3, 2)] u = 4, u.d = 1

v = ✗ ✗

Q = [(3, 2)] u = 7, u.d = 1

Q = [(3, 2), (9, 2)] v = ✗ ✗ 9, v.d = 2 9 is black

Q = [(9, 2)] u = 3, u.d = 2

Q = [(9, 2), (6, 3)] v = ✗ ✗ ✗ 6, v.d = 3

6 is black

Q = [(6, 3)] u = 9, u.d = 2

Q = [(6, 3), (8, 3)] v = ✗ 8, v.d = 3 8 is black

Q = [(8, 3)] u = 6, u.d = 3

Q = [(8, 3), (5, 4)] v = 5, v.d = 4 5 is black

v = ✗ ✗

Q = [(5, 4)] u = 8, u.d = 3

v = ✗ ✗

Q = [ ] u = 5, u.d = 4

v = ✗