Main Idea :-

In this Node-Labelling decision problem we have the Inputs :-

1. An undirected graph $G = (V, E)$
2. A set of k labels $K = \{0, 1, \ldots k-1\}$, where $k \leq |V|$
3. A non negative integer R.

We have to assign labels to nodes such that for every node $v \in V$, $r(v)$ is as small as possible. Since $m(v)$ is a lower bound of $r(v)$, we would like the ratio $r(v)/m(v)$ to be as small as possible.

So, we have to output a valid labelling that assigns a label $c(v) \in K$ to every node $v \in V$, such that the maximum ratio

$$\max_{v \in V} \frac{r(v)}{m(v)}$$

the result is optimized if the algorithm has

$$\max_{v \in V} \frac{r(v)}{m(v)} \leq \rho$$

for all possible instances, we call the algorithm a "ρ-proximity algorithm" and call ρ the "proximity ratio" of the algorithm

My algorithm uses a double BFS approach to label the nodes in the graph. The first BFS traversal starts from node 0 and labels the first k new nodes encountered with unique labels. This is to make sure that the first k values are sufficiently seperated from from each other & receive different labels.

The second traversal of BFS initiated from nodes that were not labelled in the first traversal & labels them based on their nearest

neighbors that have already been labeled. This is where the greedy approach comes into play. For every node that is unlabelled, the algorithm identifies it's nearest neighbor & assigns it the same label subject to the constraint that no more than k values should have the same label. The greedy approach makes sure that nodes with the same label are as close to each other as possible, while maintaining balance between number of nodes with each label. By using double BFS we can make sure that we can make sure that that algorithm labells all nodes in the graph, even the nodes not connected directly to node '0'. By implementing both double BFS and greedy, the algorithm is able to optimize the labeling of nodes in the graph by assigning labels that group together nodes that are similar or related to each other while maintaining balance between the number of nodes with each label.

Pseudo Code :-

```
OPT_LABEL(adj, k_val):
    bq, vi, n_cut, curr_l, l_list = queue(), set(), 0, 0, [-1 ... len(adj)]
    vi.add(0)
    bq.put(0)
    while bq.size() > 0:
        node = bq.get()
        if n_cut >= k_val:
            gq, iv, il = queue.Queue(), set(), set()
            gq.put(node)
            iv.add(node)
            while gq.size() > 0:
                ind = gq.get()
                il.add(ind)
                temp_il_len = len(il)
```

```
            if k_val == temp_i_len:
                list[node] = list[in_nd]
                break

        for i_val in adj[in_nd]:
            is_visited = False
            for v in vi:
                if v == i_val:
                    is_visited = True
                    break
            if not is_visited:
                q.put(i_val)
                vi.add(i_val)

    def:
        list[node], n_cnt, curr_l = curr_n_cnt+1,
                                    curr_l+1

    for k_val in adj[node]:
        is_visited = False
        for v in vi:
            if k_val == v:
                is_visited = True
                break
        if not is_visited:
            log.put(k_val)
            vi.add(k_val)

return list
```

Time Complexity :-

The time complexity of this algorithm is $O((V+E)^2)$, where V is the number of vertices and E is the number of edges in the input graph. The algorithm uses a double BFS approach, which visits each node and edge in the graph once. In the worst case, the first BFS traversal of the graph may visit all nodes and edges in the graph, which takes $O(V+E)$ time. For each visited

node, the algorithm performs a second BFS traversal of the graph, which can also visit all nodes and edges in the graph in the worst case. Thus, the time complexity of the double BFS part of the algorithm is $O((V+E)^2)$

In addition to the double BFS, the algorithm uses a greedy approach to assign labels to nodes. This operation is performed in constant time per node, so it doesn't significantly affect the overall time complexity of the algorithm. Therefore, the overall time complexity of the algorithm is $O((V+E)^2)$

Proof of correctness (proximity ratio)

This is to prove that the algorithm for the "Node labelling Optimization Problem" with $G = (V, E)$ being a tree has a "proximity ratio" of f, where f is a constant finite number that you specify.

Max ratio of small $A_{small} = 1$
Max ratio of medium $A_{med} = 1$
Max ratio of large $A_{large} = 1$

We initialize the labels for all nodes to -1. The algorithm implemented in the code above starts by assigning labels to the first k values nodes encountered using BFS travel of the graph starting from node 0. Each node receives a unique label, and the number of nodes assigned to each label is kept track of using a variable to count number of nodes. Once the first k value nodes have been labeled, the algorithm enters a greedy phase where it assigns labels to the remaining nodes in the graph. For each node, it looks for the nearest neighbor with the lowest label that has already been assigned, subject to the constraint that no more than k value nodes should have the same label. This greedy approach ensures

that nodes with the same label are as close to each other as possible, while also maintaining a balance between the number of nodes with each label. The algorithm continues this process until all nodes have been assigned a label, and returns the resulting label list.

$$r(v) \triangleq \min\{ n \mid |c(v,n)| = k \}$$

$r(v)$ is the smallest integer such that the node v can find all the k distinct labels within its neighborhood of radius $r(v)$

$$m(v) \triangleq \min\{ n \mid |N(v,n)| \geq k \}$$

$m(v)$ is the smallest integer such that node v has at least k nodes in its neighborhood of radius $m(v)$.

Since every node has exactly one label, we have $r(v) \geq m(v)$

Now suppose that there exists a node v such that $r(v) > m(v)$. This would mean that there exists a small integer h_1 such that the neighborhood of v contains at least k nodes, and there also exists a smaller integer $h_2 < h_1$ such that all k distinct labels can be found within the neighborhood of v upto distance h_2. However this is not possible because if the neighborhood of v contains at least k nodes up to distance h_1, then by definition, all k distinct labels must be present within that $r(v)$ cannot be greater than $m(v)$ for any node v. So, Maximum ratio of r/m is always 1 for node labelling problem. The labelling ensures that no two nodes within the k steps of each other have the same label, as function ensures that the set of k-nearest neighbor has distinct labels. Therefore the maximum ratio is always r/m.

Proof that "Node-labelling decision Problem" is NP-complete

To show that the node labelling problem is NP-complete, we can reduce the well-known NP-complete problem called the vertex colouring problem to the node labelling problem. The vertex colouring problem is defined as follows:

Given an undirected graph G_0 find the minimum number of colors required to color the vertices of G_0 such that no two adjacent vertices have the same color. We will construct a reduction from the vertex colouring problem to the node labelling problem. Let's assume we have an instance of the vertex coloring problem with an input graph G_0.

Construction :-

For each vertex in G_0 create a corresponding node in the node labelling problem instance. For each edge (u,v) in G_0 create an edge between the corresponding nodes in the node labeling problem instance. Set K value in the node labeling problem instance to be equal to the minimum number of colors required to color G_0.

Claim :- There exists a valid vertex-colouring of G_0 using K colours if & only if there exists a valid node labeling in the constructed node labelling problem instance.

Proof :-

If there exists a valid vertex coloring of G_0 using K colors :- Assign labels 1 to K to the corresponding nodes in the node labeling problem instance respecting the coloring of G_0. Since no two adjacent vertices in G_0 have the same color no two adjacent nodes in the constructed node labelling problem instance will have the same label. Therefore, this labeling is a valid solution to the node labeling problem.

If there exists a valid node labeling in the constructed node labelling problem instance, assign colors to the vertices of G based on the labels of the corresponding nodes in the node labelling problem instance. Since no two adjacent nodes in the constructed node labelling problem instance have the same label, no two adjacent vertices in G will have the same color. Therefore this colouring is a valid solution for the Vertex coloring problem.

The reduction from the Vertex-coloring problem to the node labelling problem shows that the node labelling problem is atleast as hard as the Vertex coloring problem, which is known to be NP-complete. Therefore, the node-labelling problem is also NP-complete.