Node-Labelling Decision Problem

UIN: 232009024

We need to prove that given a graph $G=(V,E)$, a set of K labels and a non-negative integer R such that $|C(v,R)|=K$, ~~and a set of labels it is possible or not~~ Node-Labling decision problem is NP-complete.

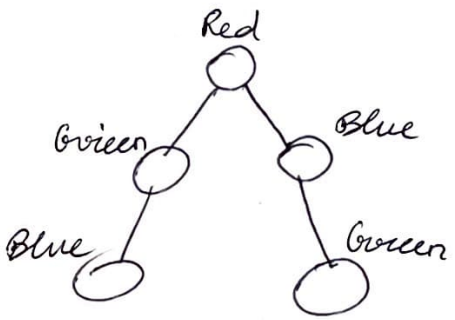
We can proof it as N-P complete by proving two things :-

1. Given a solution for an instance in problem we can verify in polynomial time that solution is correct. i.e it belongs to NP
2. For a problem that is NP complete is reducible to above problem in polynomial time - i.e ~~it belongs to~~ NPC problem is reducible to it.

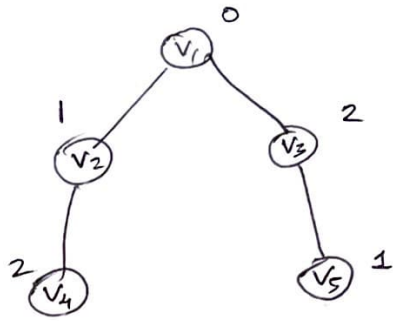
Let us assume our problem as P.

Proof(1): We need to check for every node using BFS for given R if there exist a labeling that assigns a label $c(v) \in K$ to every node $v \in V$, such that for every node we have $|C(v,R)|=K$ within R. We can do this using BFS at every node. The time complexity for this will be polynomial. Since we ~~can~~ ~~were able~~ can verify certificate in polynomial time, we can say $P \in NP$

Proof (2): Let us assume a NPC problem B. We have to proof that B is reducible to P in polynomial time. We will use the 3-coloring problem as B. Let us assume 3 colors as Red, Green and Blue in the 3 coloring problem



G(V,E)



G'(V', E')

G(V,E) be the 3 coloring graph. We will create a new graph G'(V',E') such that every node of G corresponds to every node of G' and every edge in G corresponds to G' such that V' = V and E' = E.
Red as label 0, green as 1 and Blue as label 2. Using this we created G' graph from G in polynomial time just by using BFS. We can say that B is reducible to Node-labelling problem in polynomial time.

We can show reverse case as well.
Let us create a graph G(V, E) 3-coloring problem from Node-labelling problem G'(V, E). such that V = V' and E = E'. Label 0 as Red, label 1 as Green and label 2 as Blue. Hence we can create G from G' in polynomial time. We can see that no two adjacent vertices have same color. Hence we can see that $P \leq_P B$ i.e Node-labeling problem is reducible to 3-coloring problem in polynomial time.

Since we proved both points, we can say that Node labelling decision problem is NP complete.

Node-Labelling Optimization Problem

UIN: 232009024

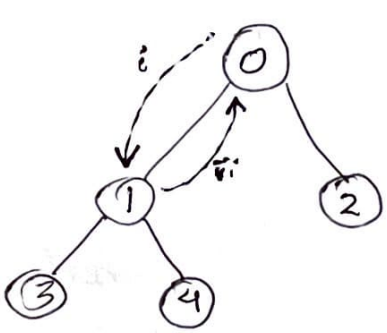
① Main Idea:

- Input: An undirected graph $G = (V, E)$
  A set of k labels $K = \{0, 1, \ldots, k-1\}$, where $k \leq |V|$
- Output: A valid labelling that assigns a label $c(v) \in K$ to every node $v \in V$, such that the maximum ratio $\max_{v \in V} \frac{r(v)}{m(v)}$ is minimized.

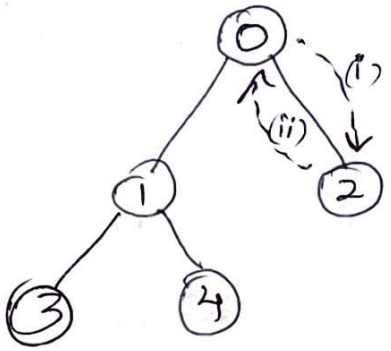
Main idea is to label the nodes based on our goal i.e to minimize $r(v)$ values. So what I am doing is I am using the label that is farthest from the given node to label itself. I have created individual dictionaries for each node that stores distance of all K labels from itself. Then, based on these distances value, node take the label that has max distance value. Initially all distance values are assigned as infinite. Then every node before assigning take updated information from root node (that I doing inside forward function) and based on updated information decides its value (give_label function) and then shares updated information back to root node (backtrack function in code).

All the nodes are traversed using BFS. And for forward and backtrack function we will travel through a predecided path that we calculate using DFS.

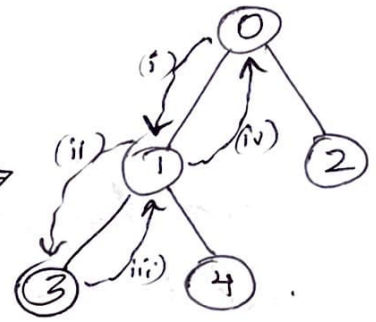
Traversal is as shown :-



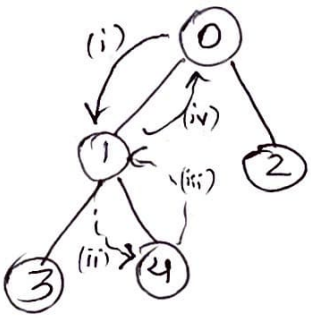
(While deciding label of node 1)



(for deciding label of node 2)



(for node 3)



(for node 4)

② Pseudo code :

```
function Node_labeler (adjlist, k)
    label=[] (array to store labels)
    seen =[] (to track visited nodes)
    parent_labels = defaultdict(dict)   { dic to store distance of kth label from node }
    assing all value for k label as infinite

function path_from_root(des, visited, path, path_storer, adjlist):
    if visited [des] == -1 :
        visited [des] = 1
        path_storer [des] = path.copy()
        for child in adjlist [des] :
            path.append(child)
            path_from_root(child, visited, path, path_storer)
            path.pop()

    path_storer ={}
    visited = [-1] * len(adjlist)
    path_from root (0, visited, [0], path_storer, adjlist)

function give_label(i):
    - sort parent_labels [i] dictionary in reverse
      order based on values.
    - label [i] = sorted_dict [0]
      parent_labels [i] [sorted_dict [0]] = 0
      return
```

function backtrack(j):

for i in range(len(path_storer[j])-1, 0, -1):

compare dictionary of destination(i-1) with source(i) and update distance values.

{ if destin(K^th value) > source(K^th value):

destin(K^th value) = source(K^th value) + 1

return

function forward(j):

for i in range(len(path_storer[j])-1):

compare dictionary of destination(i+1) with source(i) and update distance using same logic as of backtrack.

return.

```
q = deque()
q.append(0)
seen[0] = 1
label[0] = 0
parent_labels[0][0] = 0
while q:
    idx = q.popleft()
    for j in adj_list[idx]:
        if seen[j] == -1
            q.append(j)
            seen[j] = 1
```

```
        forward(i)
        give_label(i)
        backtrack(i)

    return label

function main ("filename_of_trees", "filename of K",
                "filename for sol"):

    adjtree_list = (load trees)
    K_values_list = (load K values)

    solution = []
    for i in range (len.(adjtree_list)):
        labels = Node_labeler (adjtree_list[i],
                               K_values_list[i])

        solution.append(labels)

    pickle.dump (solution, "sol" filename)
```

---

③ Time Complexity :-

Node_labeller { ⟶ BFS ⇒ O(V+E)

Forward ⟶ max H height & updatedic ⇒ O(KH)
give label ⟶ comparing dictionary K values ⇒ O(K)
Backtrack → max H height & update dic ⇒ O(KH)

Path_from_root ⟶ DFS ⇒ O(V+E)

Overall time complexity $= \underset{DFS}{O(V+E)} + \underset{BFS}{O\left(V(KH+K+K+1)+E\right)}$

$= O\left(V+E+V*(2H+1)*K+E\right)$

$= \underbrace{O\left(2E+V*\left(1+K*(2H+1)\right)\right)}_{\text{polynomial.}}$

(4) Analysis of Algorithm

⇒ We are assigning labels $c(v) \in K$ to every node in the graph with the goal to minimize proximity ratio. i.e $\frac{r(v)}{m(v)}$.

$m(v)$ is the smallest integer such that the node v has at least K nodes in its neighbourhood of radius $m(v)$. (including itself). So we could ~~think~~ ~~$m(v)$~~ say $m(v)$ will contain minimum of K nodes.

⇒ We will show that our algorithm will place all distinct K labels in m hops.

$\therefore$ proximity ratio $= \frac{r}{m} = \frac{m}{m} = 1$

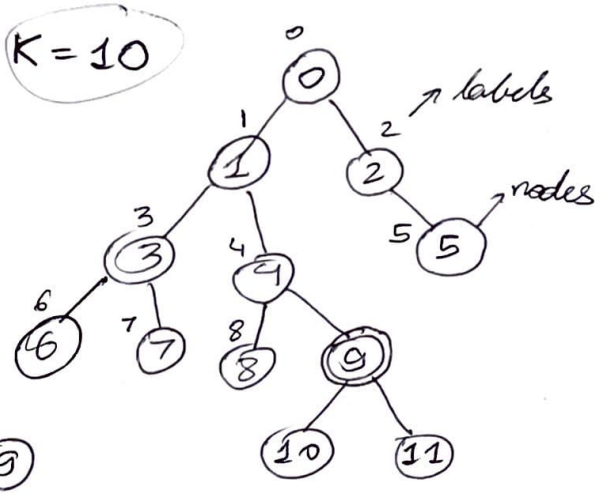
We are traversing the graph using BFS so we will move from root nodes towards bottom. So every node while assigning label will have information of distance of all the nodes ~~above~~ top of its level. It will then calculate the label that has ~~so~~ largest distance and assign it to self. — By this it is bringing the farthest label inside m-hops from it.

The childs of this node will get updated information from it and they will again label themselves such that the label that ~~will~~ were farthest from parent will come inside m hops from parent.

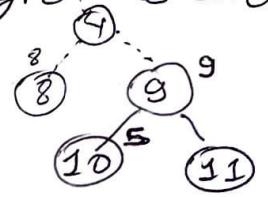
As stated earlier, since we know that m hops will have min K nodes for every nodes. So ~~doo~~ using above algorithm all K labels will come under m-hops.

~~[illegible]~~

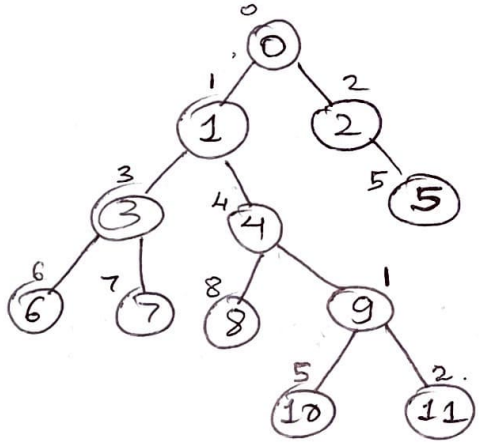
Let us try to understand this using an example. Let say at point when we are deciding the label for node (9) in below example.



we know for node ⑨
$m = 4$. Node ⑨ can find all the label except 5 and 9 ~~Bestad~~ in m hops. It could find 5 in 5 hops but it cannot find 9. So node ⑨ will assign its label as 9.
Now all the label for ⑨ are inside m-hops except 5. So its child will assign its label as 5 to bring it close to ⑨.
And hence prr ratio will become 1.



⇒ We can proof that every node can find K-distinct labels in m-hops using contradiction.
Let us assume that all K labels are not present in m-hops from node v. This means node v will have label that already exist within m hop. and there is a missing label outside m hop.
But as per our algorithm we are labeling based on farthest distance data. So this can never be a possible scenario. There is contradiction.



For example if in this graph ~~the label of~~ for node ⑨ if the label would have been 1. We are not getting label 9 in m_hops. We can see the distance of label 9 would be ∞ and of 1 would have been 2 while ⑨ was assigning its label ~~as~~ using our algo. Our algo would have given ⑨ label as 9. and calculated ~~prox~~ ratio as 1.

This contradiction shows that always we will be able to find K-distinct labels in m hops from any node. i.e $K = m$.

Therefore prox ratio $= \frac{K}{m} = 1$

Hence Proved.