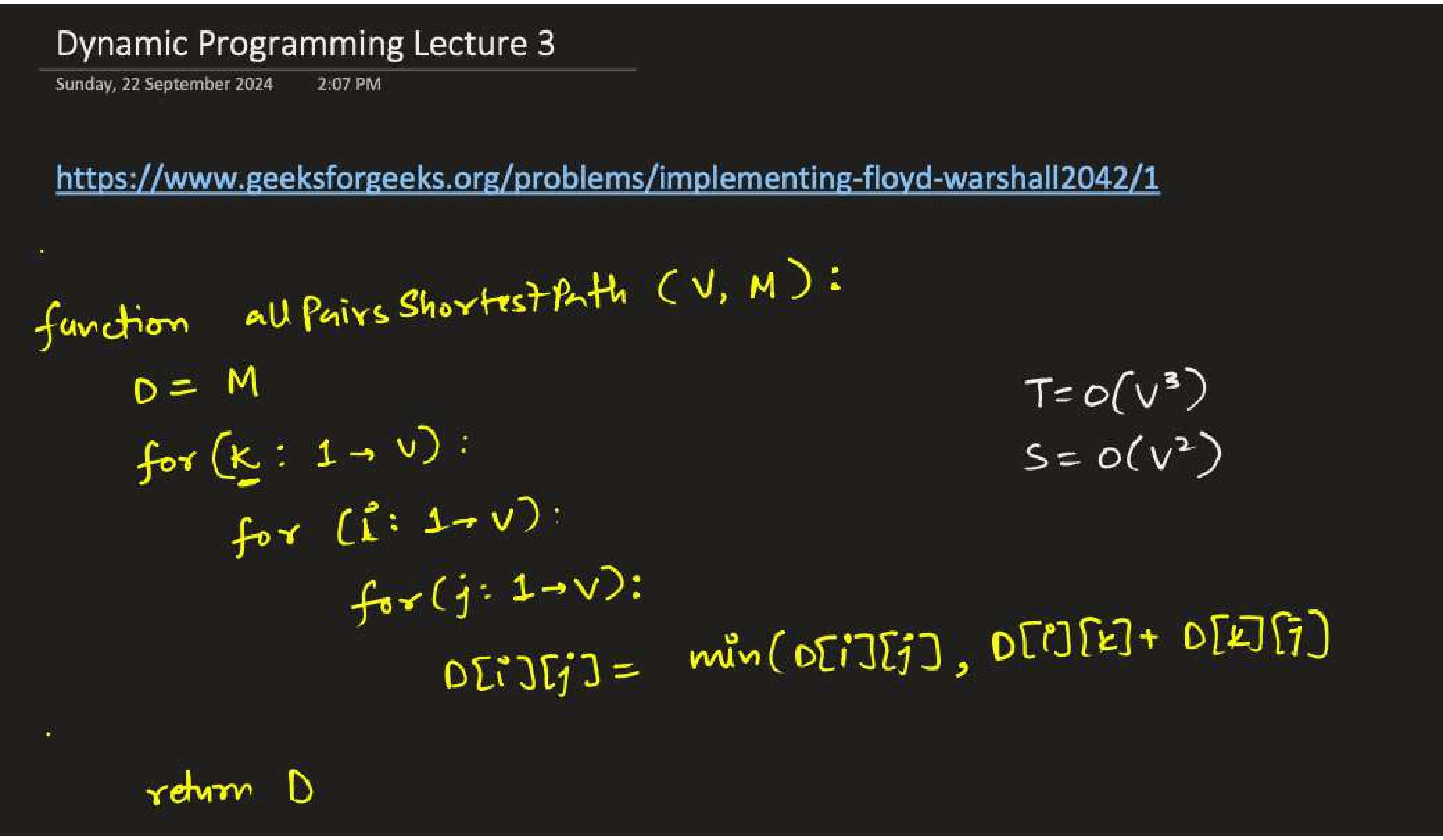


Dynamic Programming Lecture 3

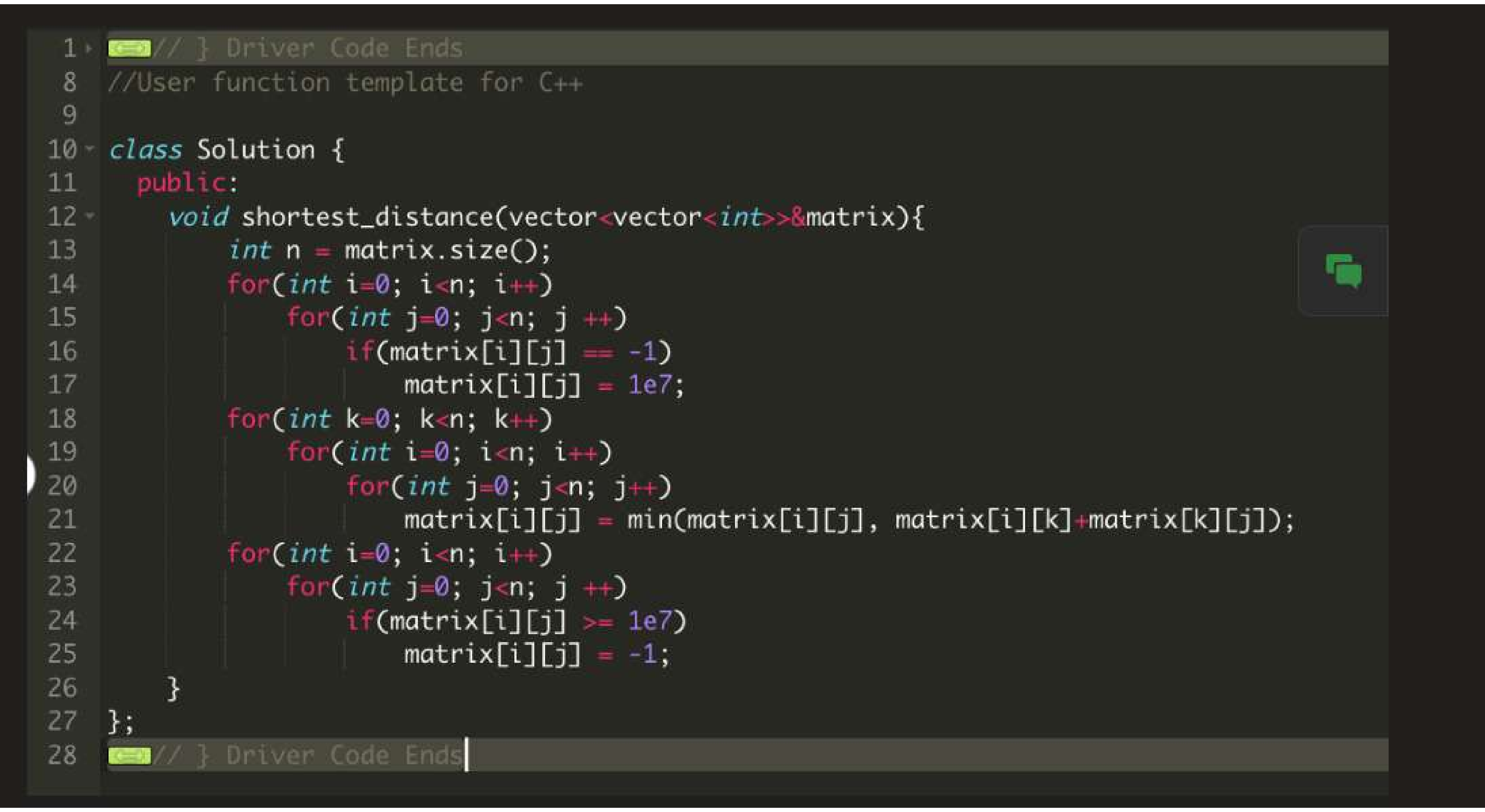
Sunday, 22 September 2024 2:07 PM

https://www.geeksforgeeks.org/problems/implementing-floyd-warshall2042/1

```
function allPairsShortestPath (V, M):
    D = M
    for (k : 1 → v):
        for (i : 1 → v):
            for (j : 1 → v):
                D[i][j] = min(D[i][j], D[i][k] + D[k][j])

    return D
```

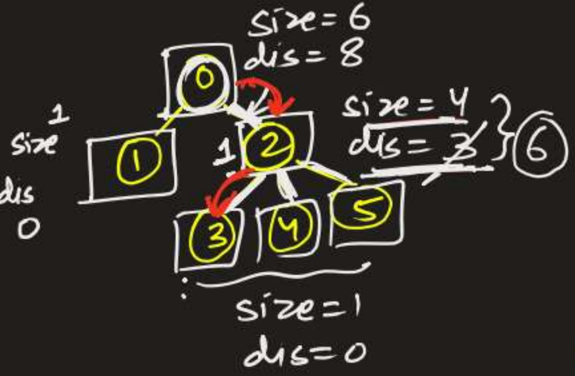
$T = O(V^3)$

$S = O(V^2)$



```
// } Driver Code Ends
//User function template for C++

class Solution {
  public:
    void shortest_distance(vector<vector<int>>&matrix){
        int n = matrix.size();
        for(int i=0; i<n; i++)
            for(int j=0; j<n; j ++)
                if(matrix[i][j] == -1)
                    matrix[i][j] = 1e7;
        for(int k=0; k<n; k++)
            for(int i=0; i<n; i++)
                for(int j=0; j<n; j++)
                    matrix[i][j] = min(matrix[i][j], matrix[i][k]+matrix[k][j]);
        for(int i=0; i<n; i++)
            for(int j=0; j<n; j ++)
                if(matrix[i][j] >= 1e7)
                    matrix[i][j] = -1;
    }
};
// } Driver Code Ends
```

https://leetcode.com/problems/sum-of-distances-in-tree/



0:-
1→1
2→1
3→2
4→2
5→2
} ⑧

1:-
0→1
2→2
3→3
4→3
5→3
} ⑫

2:- ⑥, 3:→⑩, 4→⑩ 5:→⑩

dis(0)→ ① $\boxed{1+0}$ ② 4+3

dis[node]=0, size[node]=1

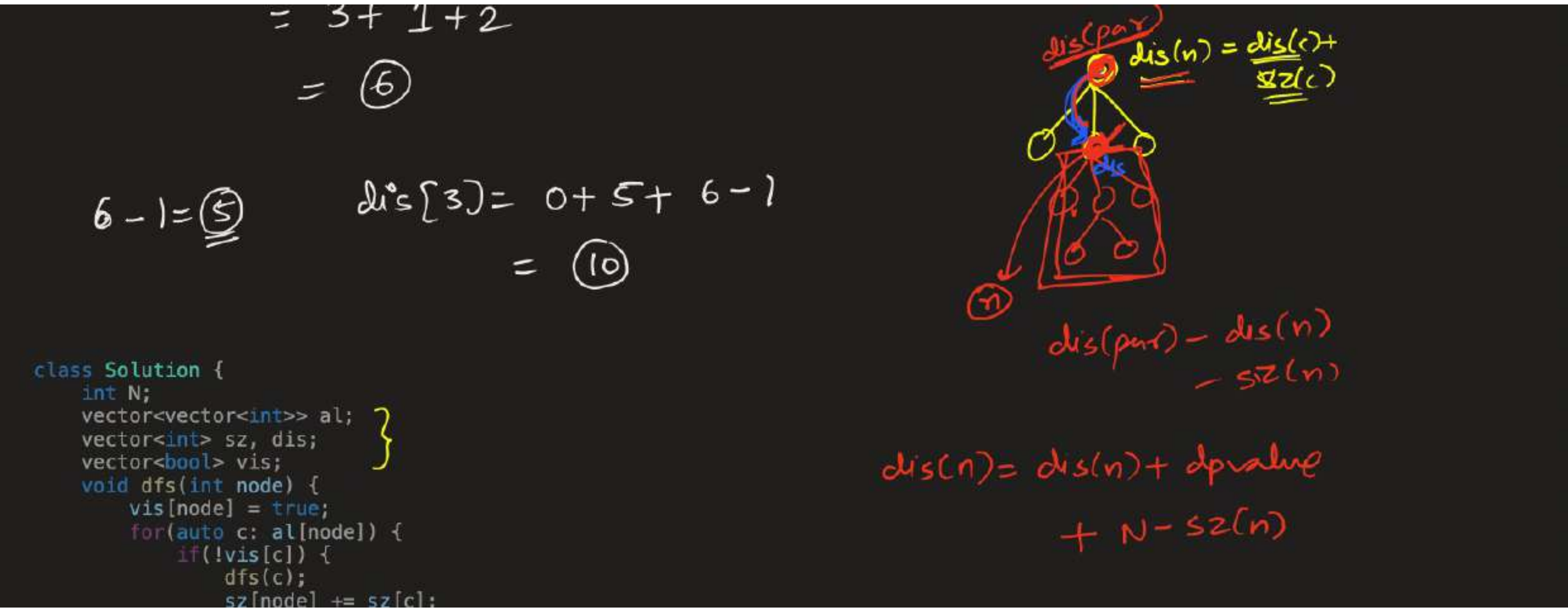
```
for child of node:-
    dis[node] += (size(child) + dis(child))
    size[node] += size(child)
```

$$\text{dis}[2) = \text{dis}[2] + \overset{\overset{1}{\text{dis}(0)}}{\text{dpVal}} + \underbrace{\overset{\text{size}(0)}{\text{sizeof tree}} - \text{size}[2)}$$

$$= 3 + 1 + 2$$

$$= ⑥$$

dis(par)
dis(n)



$= 3 + 1 + 2$

$= \textcircled{6}$

$6 - 1 = \textcircled{5}$

$dis[3] = 0 + 5 + 6 - 1$

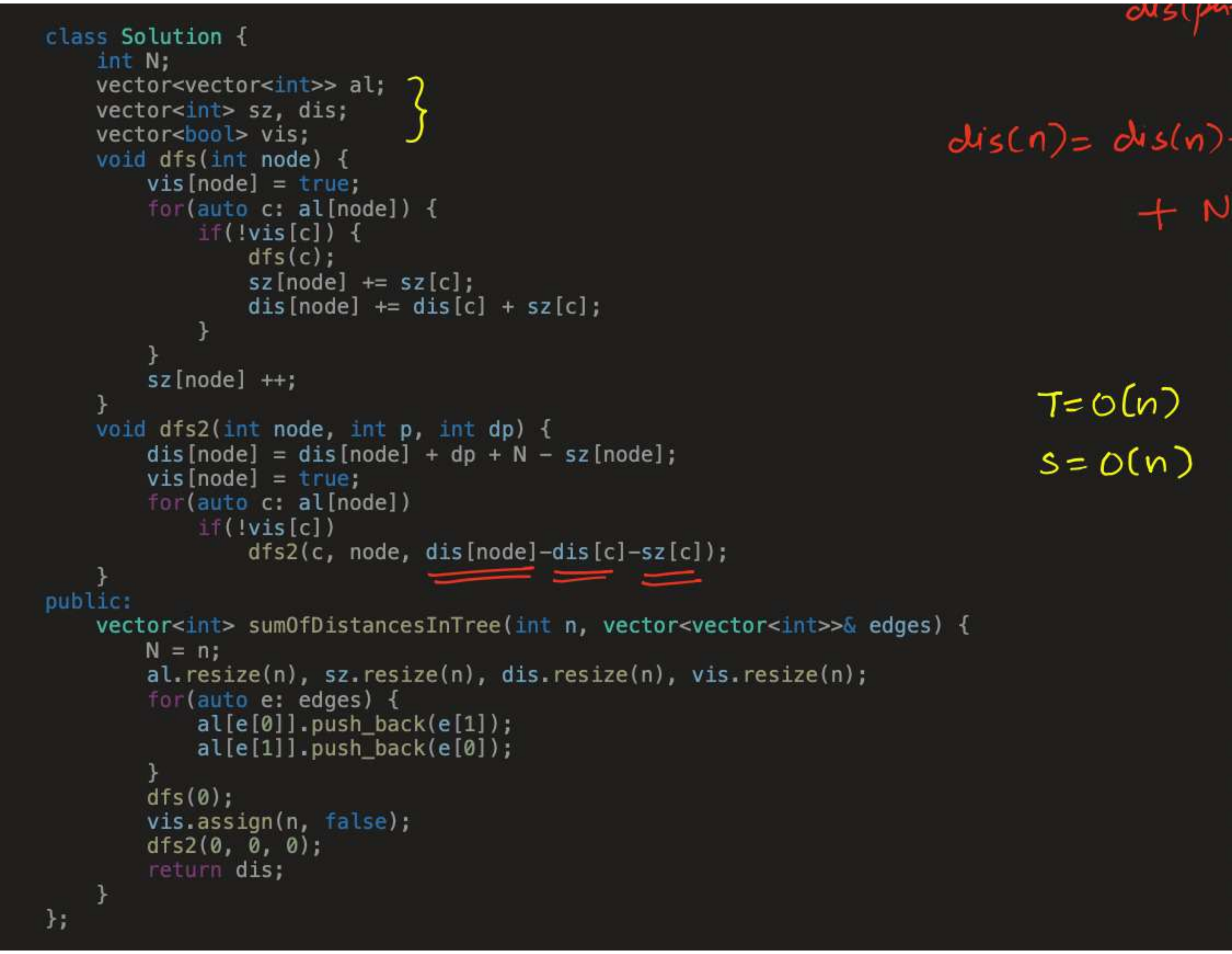
$= \textcircled{10}$

$dis(n) = dis(c) + sz(c)$

$dis(par) - dis(n) - sz(n)$

$dis(n) = dis(n) + dpvalue + N - sz(n)$

```
class Solution {
    int N;
    vector<vector<int>> al;
    vector<int> sz, dis;
    vector<bool> vis;
    void dfs(int node) {
        vis[node] = true;
        for(auto c: al[node]) {
            if(!vis[c]) {
                dfs(c);
                sz[node] += sz[c];
```



```cpp
class Solution {
    int N;
    vector<vector<int>> al;
    vector<int> sz, dis;
    vector<bool> vis;
    void dfs(int node) {
        vis[node] = true;
        for(auto c: al[node]) {
            if(!vis[c]) {
                dfs(c);
                sz[node] += sz[c];
                dis[node] += dis[c] + sz[c];
            }
        }
        sz[node] ++;
    }
    void dfs2(int node, int p, int dp) {
        dis[node] = dis[node] + dp + N - sz[node];
        vis[node] = true;
        for(auto c: al[node])
            if(!vis[c])
                dfs2(c, node, dis[node]-dis[c]-sz[c]);
    }
public:
    vector<int> sumOfDistancesInTree(int n, vector<vector<int>>& edges) {
        N = n;
        al.resize(n), sz.resize(n), dis.resize(n), vis.resize(n);
        for(auto e: edges) {
            al[e[0]].push_back(e[1]);
            al[e[1]].push_back(e[0]);
        }
        dfs(0);
        vis.assign(n, false);
        dfs2(0, 0, 0);
        return dis;
    }
};
```

dis(pa

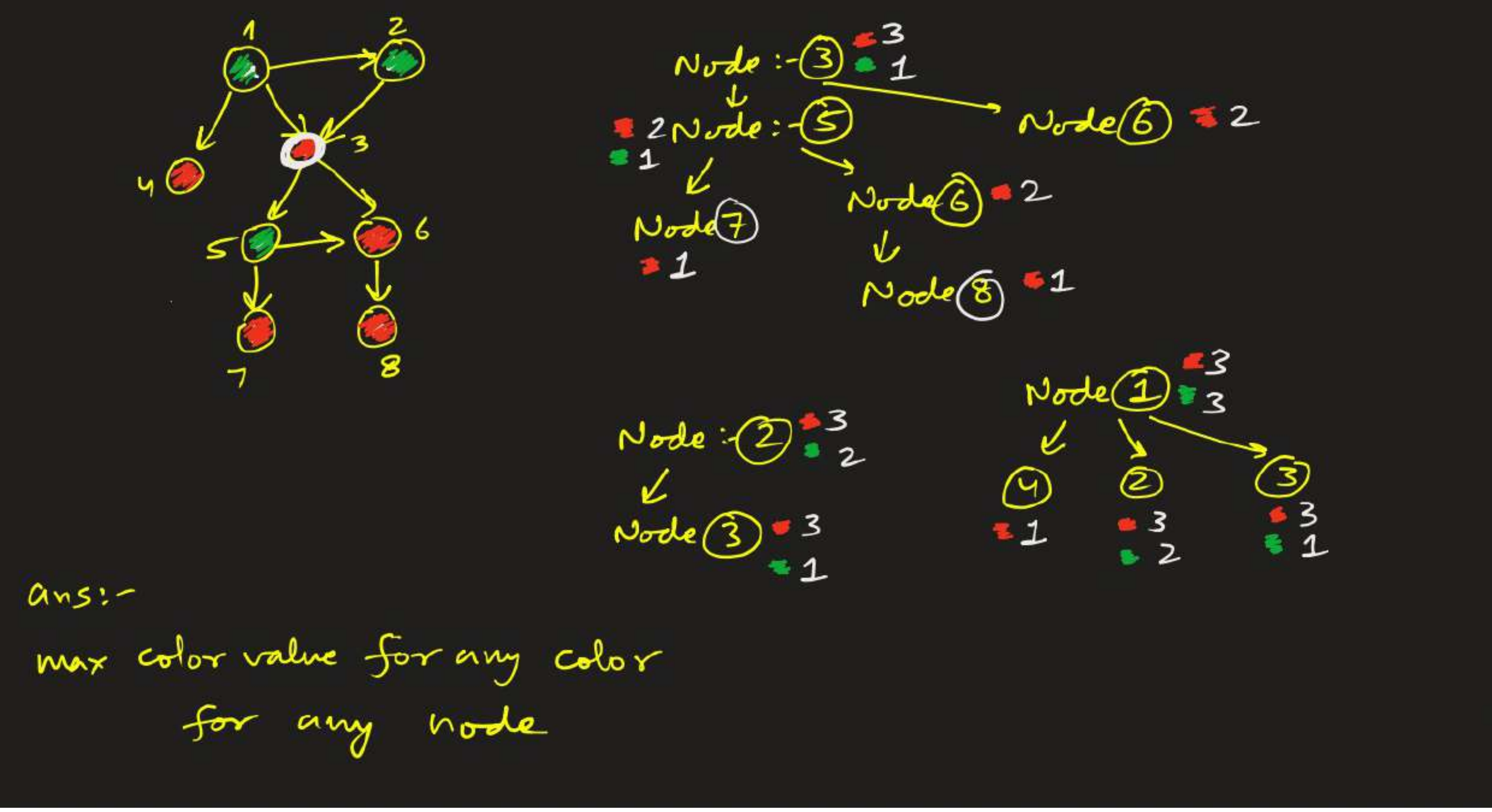
dis(n)= dis(n)-

+ N

T=O(n)

S=O(n)

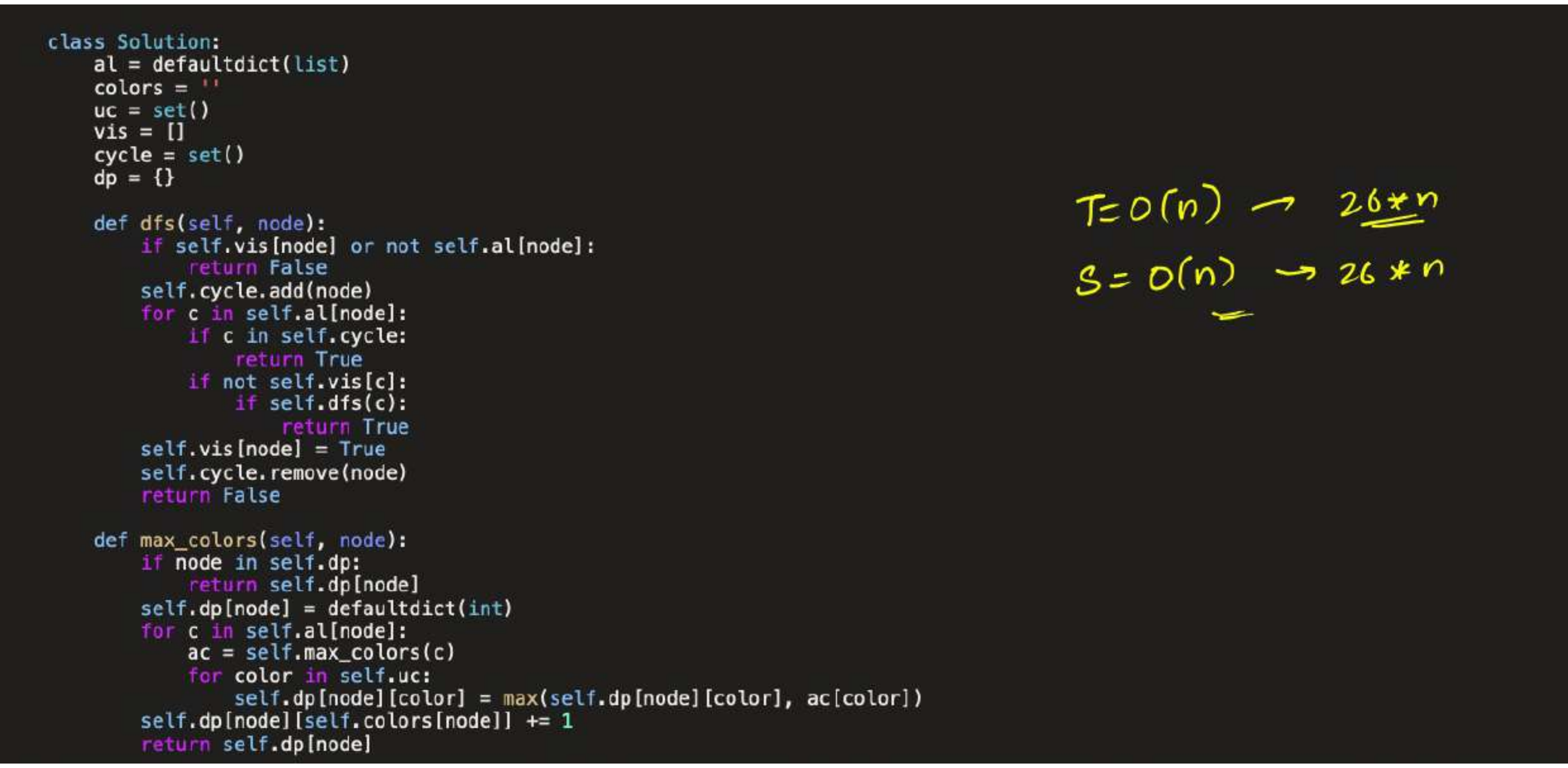
https://leetcode.com/problems/largest-color-value-in-a-directed-graph/



ans:-

max color value for any color

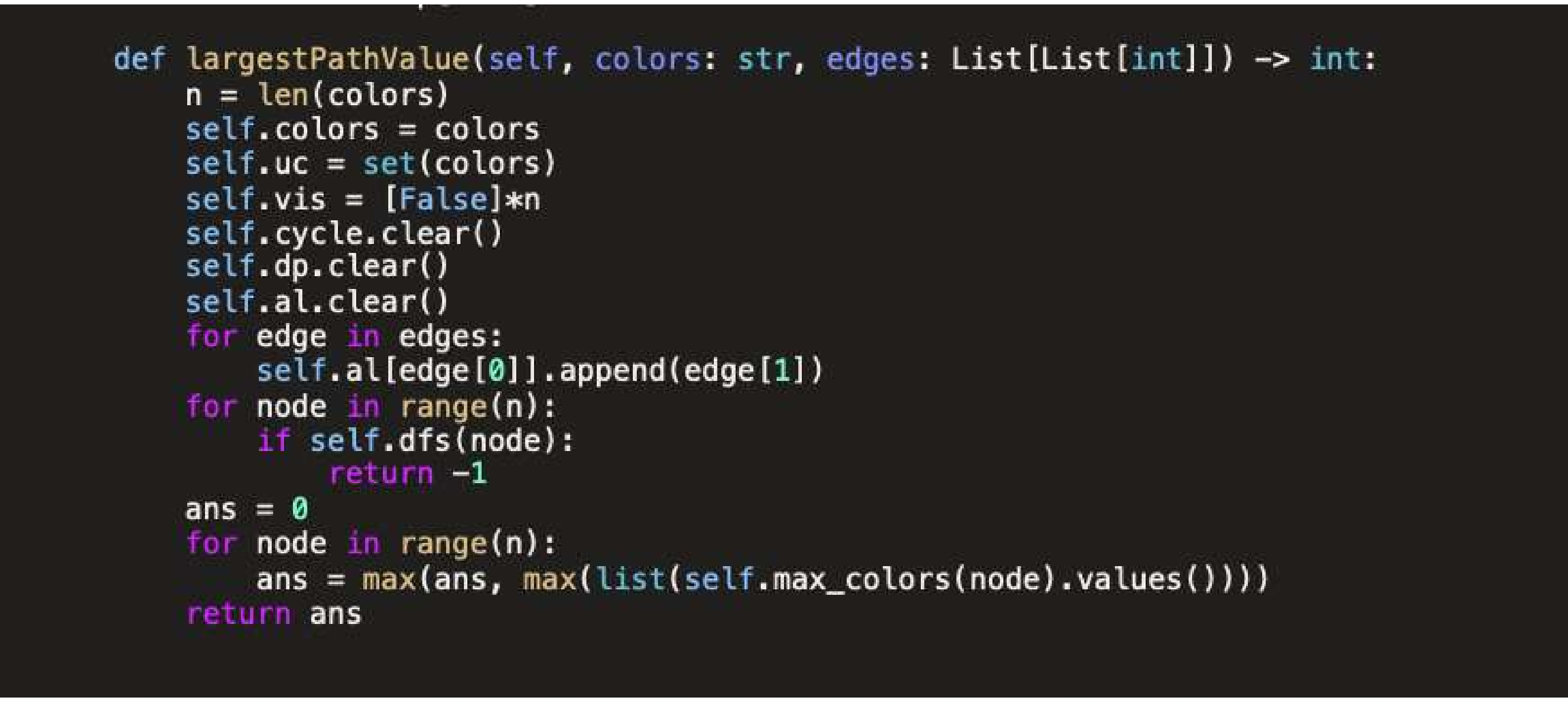
for any node



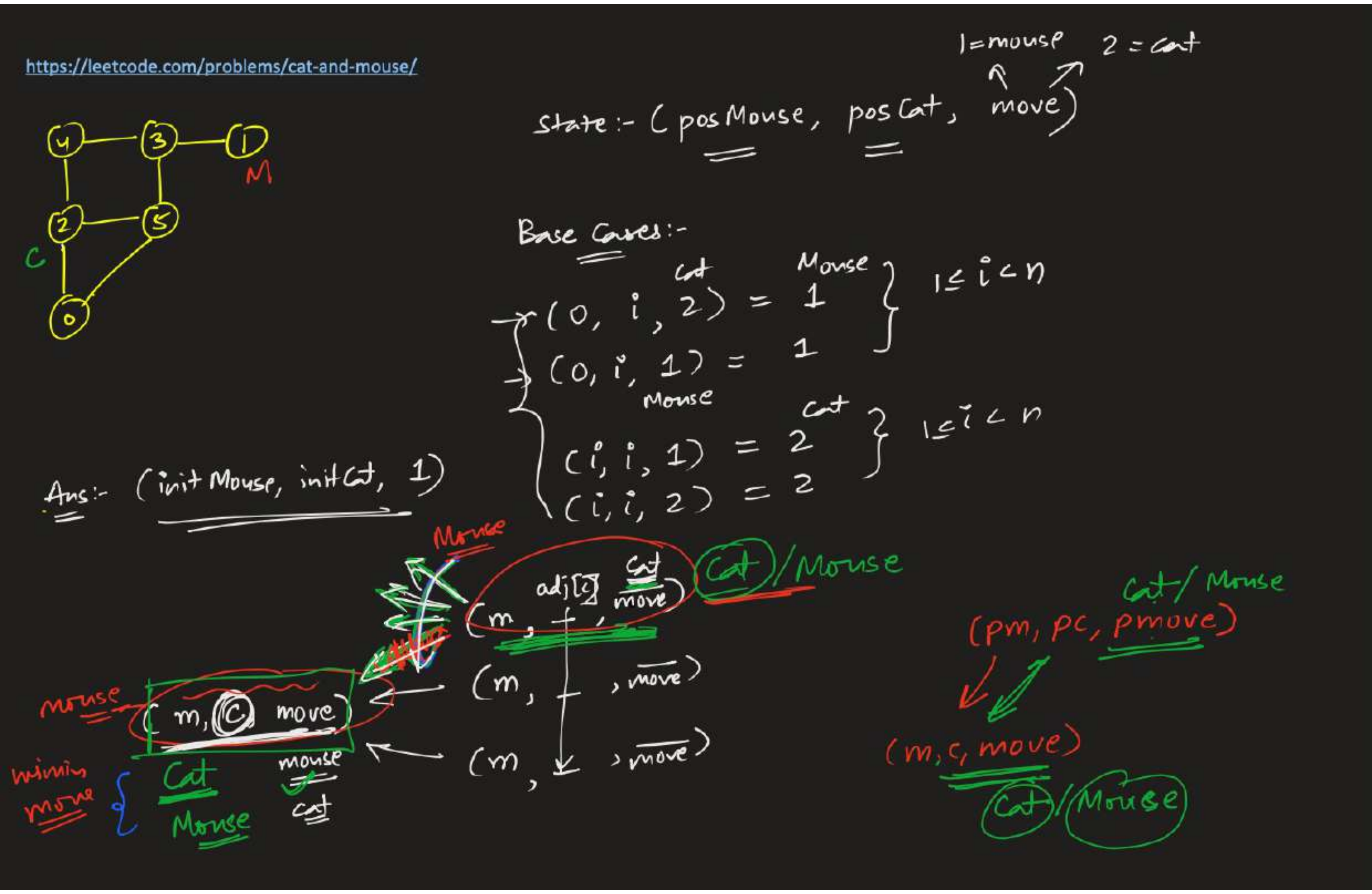
```python
class Solution:
    al = defaultdict(list)
    colors = ''
    uc = set()
    vis = []
    cycle = set()
    dp = {}

    def dfs(self, node):
        if self.vis[node] or not self.al[node]:
            return False
        self.cycle.add(node)
        for c in self.al[node]:
            if c in self.cycle:
                return True
            if not self.vis[c]:
                if self.dfs(c):
                    return True
        self.vis[node] = True
        self.cycle.remove(node)
        return False

    def max_colors(self, node):
        if node in self.dp:
            return self.dp[node]
        self.dp[node] = defaultdict(int)
        for c in self.al[node]:
            ac = self.max_colors(c)
            for color in self.uc:
                self.dp[node][color] = max(self.dp[node][color], ac[color])
        self.dp[node][self.colors[node]] += 1
        return self.dp[node]
```

$T = O(n) \rightarrow \underline{26 * n}$

$S = \underline{O(n)} \rightarrow 26 * n$



```python
def largestPathValue(self, colors: str, edges: List[List[int]]) -> int:
    n = len(colors)
    self.colors = colors
    self.uc = set(colors)
    self.vis = [False]*n
    self.cycle.clear()
    self.dp.clear()
    self.al.clear()
    for edge in edges:
        self.al[edge[0]].append(edge[1])
    for node in range(n):
        if self.dfs(node):
            return -1
    ans = 0
    for node in range(n):
        ans = max(ans, max(list(self.max_colors(node).values())))
    return ans
```



https://leetcode.com/problems/cat-and-mouse/

1=mouse 2 = cat

State :- (posMouse, posCat, move)

Base Cases :-

$(0, i, 2) = 1$ — cat, Mouse

$(0, i, 1) = 1$ — Mouse

$1 \le i < n$

$(i, i, 1) = 2$ — cat

$(i, i, 2) = 2$

$1 \le i < n$

Ans :- (initMouse, initCat, 1)

Mouse

(m, adj[c], $\overline{\text{move}}$) — cat

(Cat)/Mouse

(m, , $\overline{\text{move}}$)

(m, , $\overline{\text{move}}$)

mouse (m, (c) move)

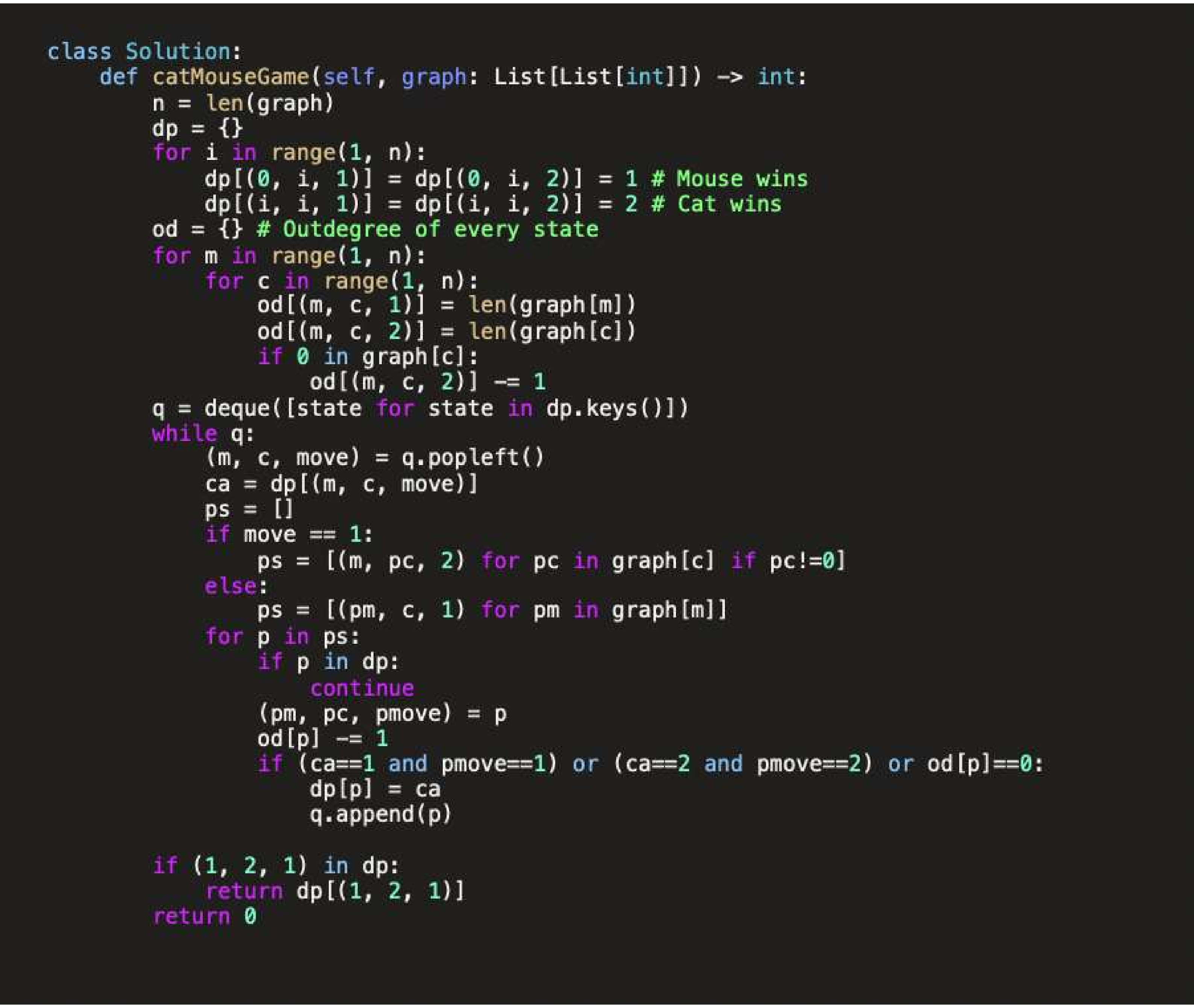
winning move { Cat / Mouse — mouse / cat

Cat/ Mouse

(pm, pc, pmove)

(m, c, move)

(Cat)/(Mouse)



```
class Solution:
    def catMouseGame(self, graph: List[List[int]]) -> int:
        n = len(graph)
        dp = {}
        for i in range(1, n):
            dp[(0, i, 1)] = dp[(0, i, 2)] = 1 # Mouse wins
            dp[(i, i, 1)] = dp[(i, i, 2)] = 2 # Cat wins
        od = {} # Outdegree of every state
        for m in range(1, n):
            for c in range(1, n):
                od[(m, c, 1)] = len(graph[m])
                od[(m, c, 2)] = len(graph[c])
                if 0 in graph[c]:
                    od[(m, c, 2)] -= 1
        q = deque([state for state in dp.keys()])
        while q:
            (m, c, move) = q.popleft()
            ca = dp[(m, c, move)]
            ps = []
            if move == 1:
                ps = [(m, pc, 2) for pc in graph[c] if pc!=0]
            else:
                ps = [(pm, c, 1) for pm in graph[m]]
            for p in ps:
                if p in dp:
                    continue
                (pm, pc, pmove) = p
                od[p] -= 1
                if (ca==1 and pmove==1) or (ca==2 and pmove==2) or od[p]==0:
                    dp[p] = ca
                    q.append(p)

        if (1, 2, 1) in dp:
            return dp[(1, 2, 1)]
        return 0
```