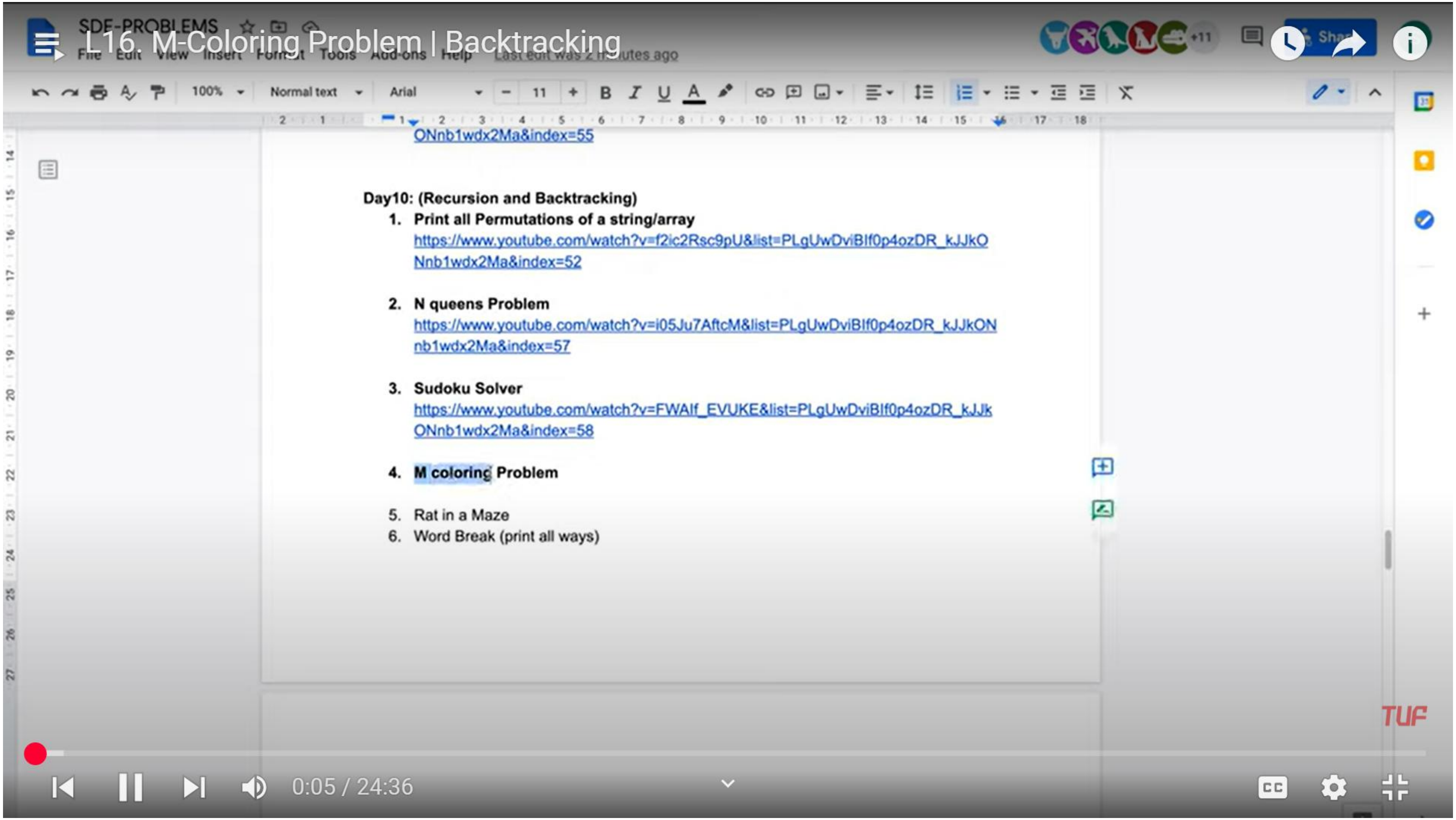


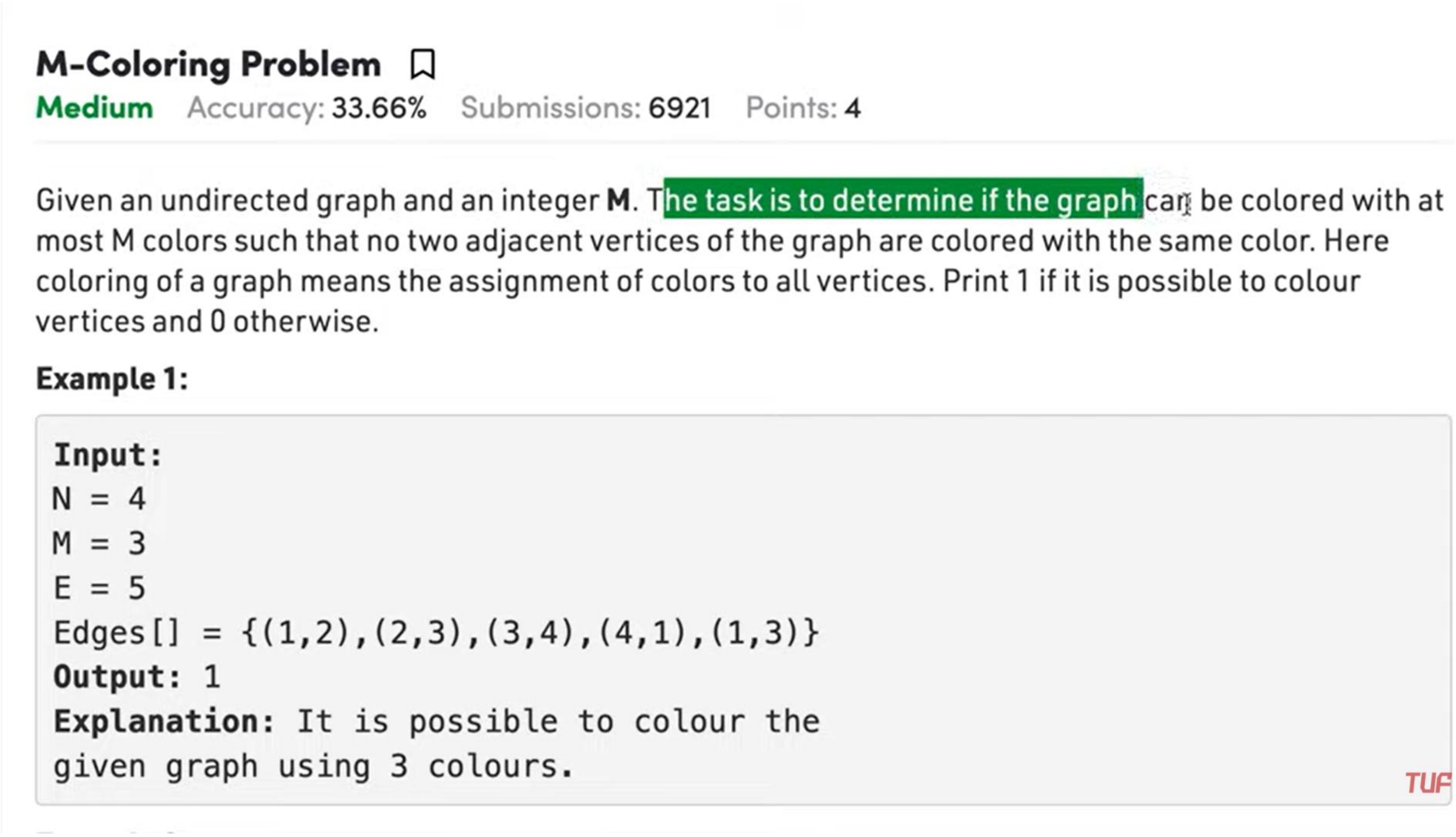
L16. M-Coloring Problem | Backtracking

SDE-PROBLEMS
File Edit View Insert Format Tools Add-ons Help Last edit was 2 minutes ago
100% Normal text Arial 11
TUF
0:05 / 24:36

ONnb1wdx2Ma&index=55

Day10: (Recursion and Backtracking)

1. **Print all Permutations of a string/array**
   https://www.youtube.com/watch?v=f2ic2Rsc9pU&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=52
2. **N queens Problem**
   https://www.youtube.com/watch?v=i05Ju7AftcM&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=57
3. **Sudoku Solver**
   https://www.youtube.com/watch?v=FWAIf_EVUKE&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=58
4. **M coloring Problem**
5. Rat in a Maze
6. Word Break (print all ways)



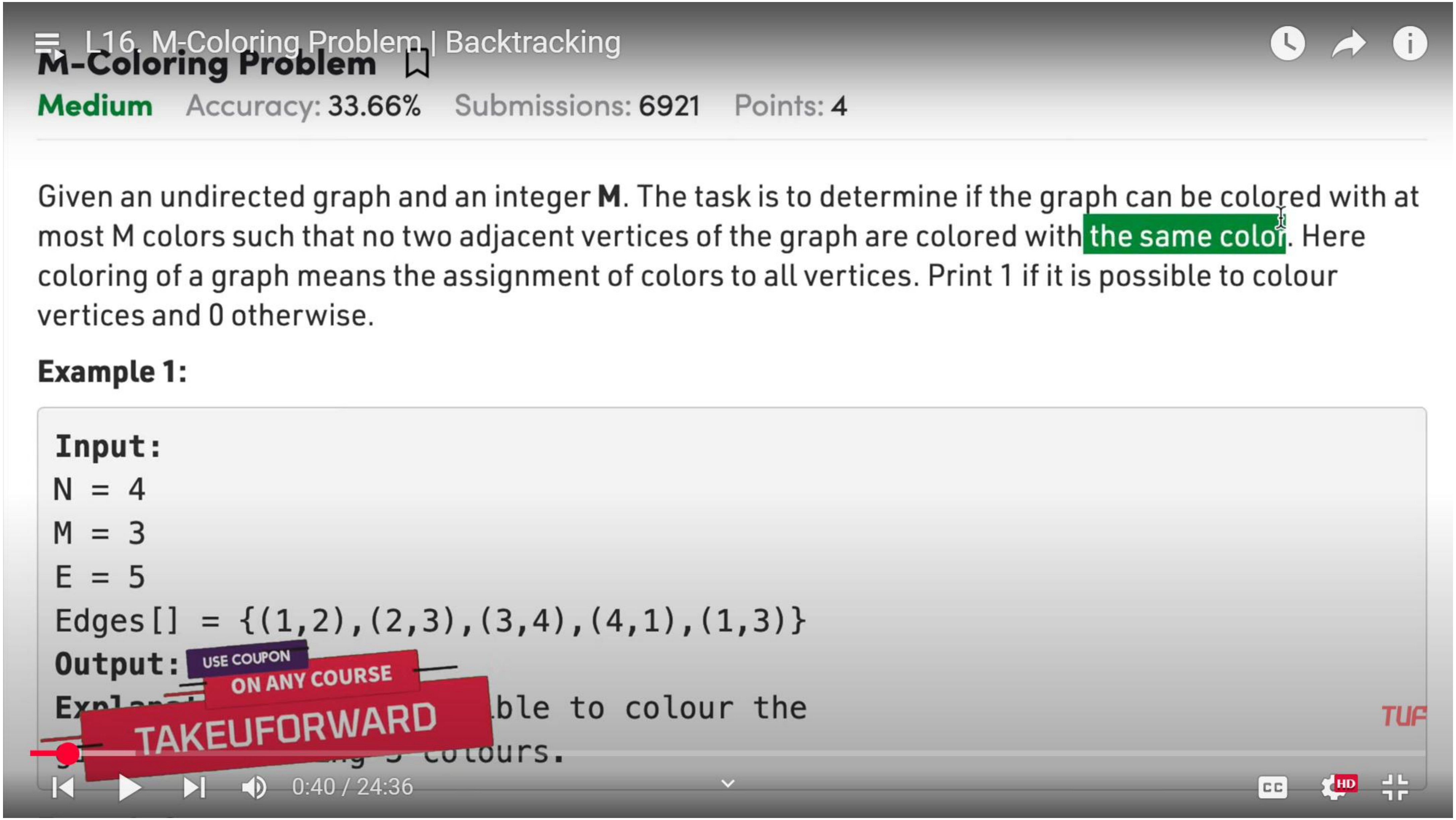
M-Coloring Problem

Medium Accuracy: 33.66% Submissions: 6921 Points: 4

Given an undirected graph and an integer **M**. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:

```
Input:
N = 4
M = 3
E = 5
Edges[] = {(1,2),(2,3),(3,4),(4,1),(1,3)}
Output: 1
Explanation: It is possible to colour the
given graph using 3 colours.
```



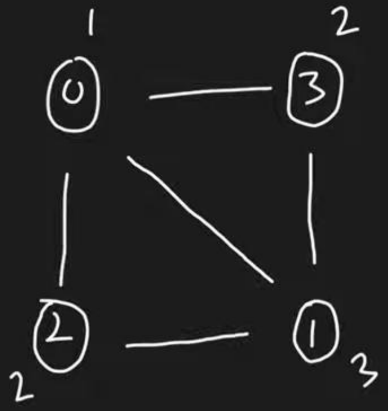
M-Coloring Problem

Medium Accuracy: 33.66% Submissions: 6921 Points: 4

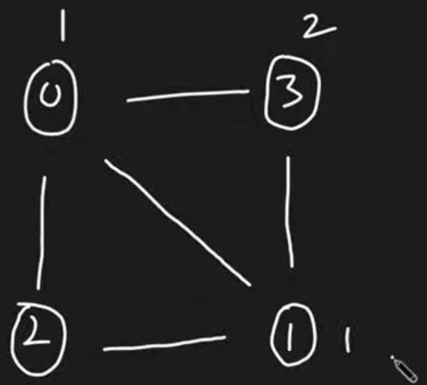
Given an undirected graph and an integer **M**. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:

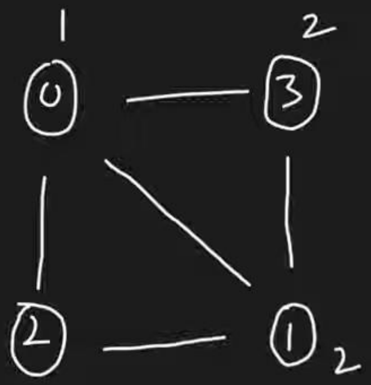
```
Input:
N = 4
M = 3
E = 5
Edges[] = {(1,2),(2,3),(3,4),(4,1),(1,3)}
Output:
Ex                  ble to colour the
                      colours.
```



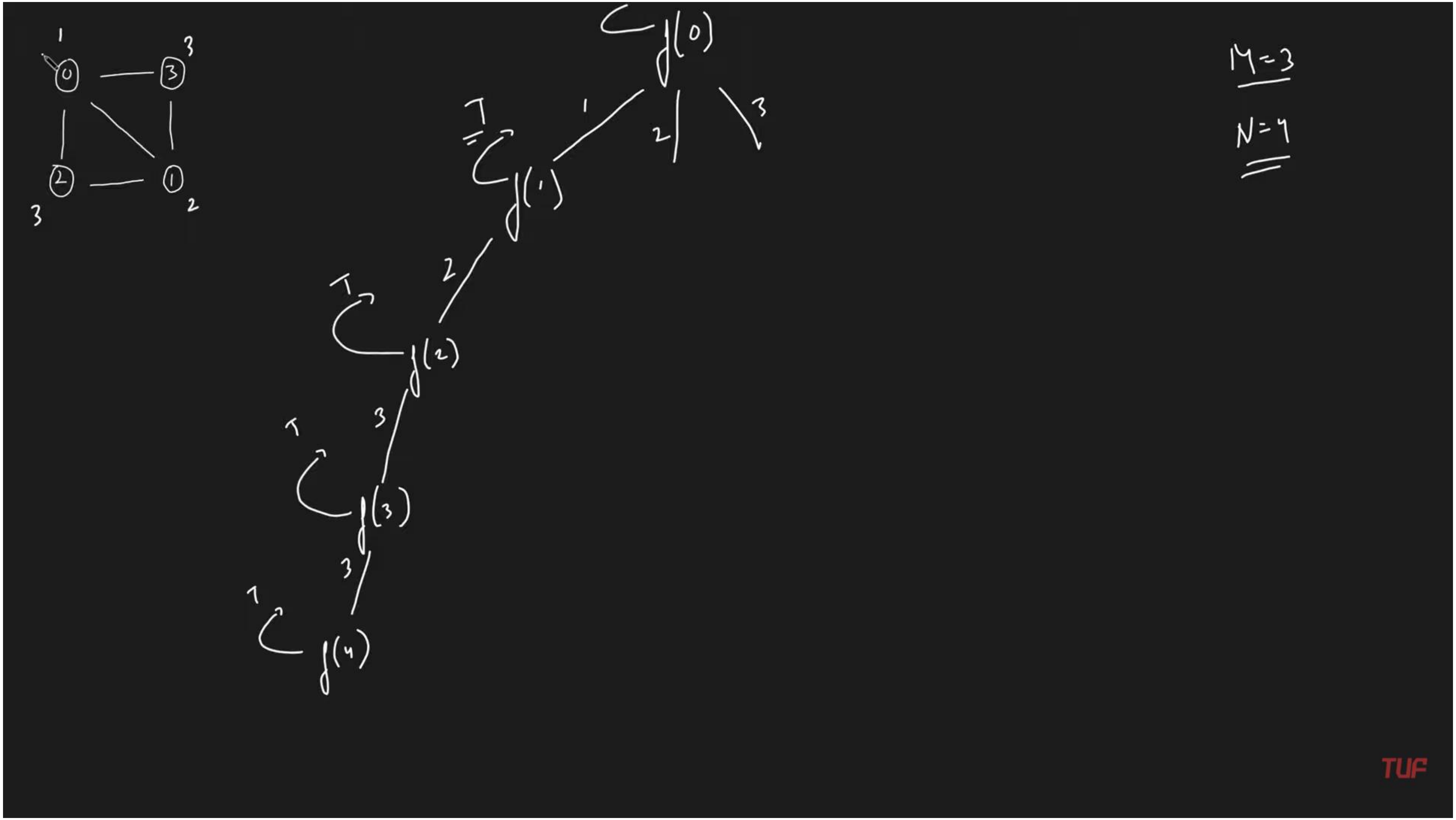
M = 3



M = 2

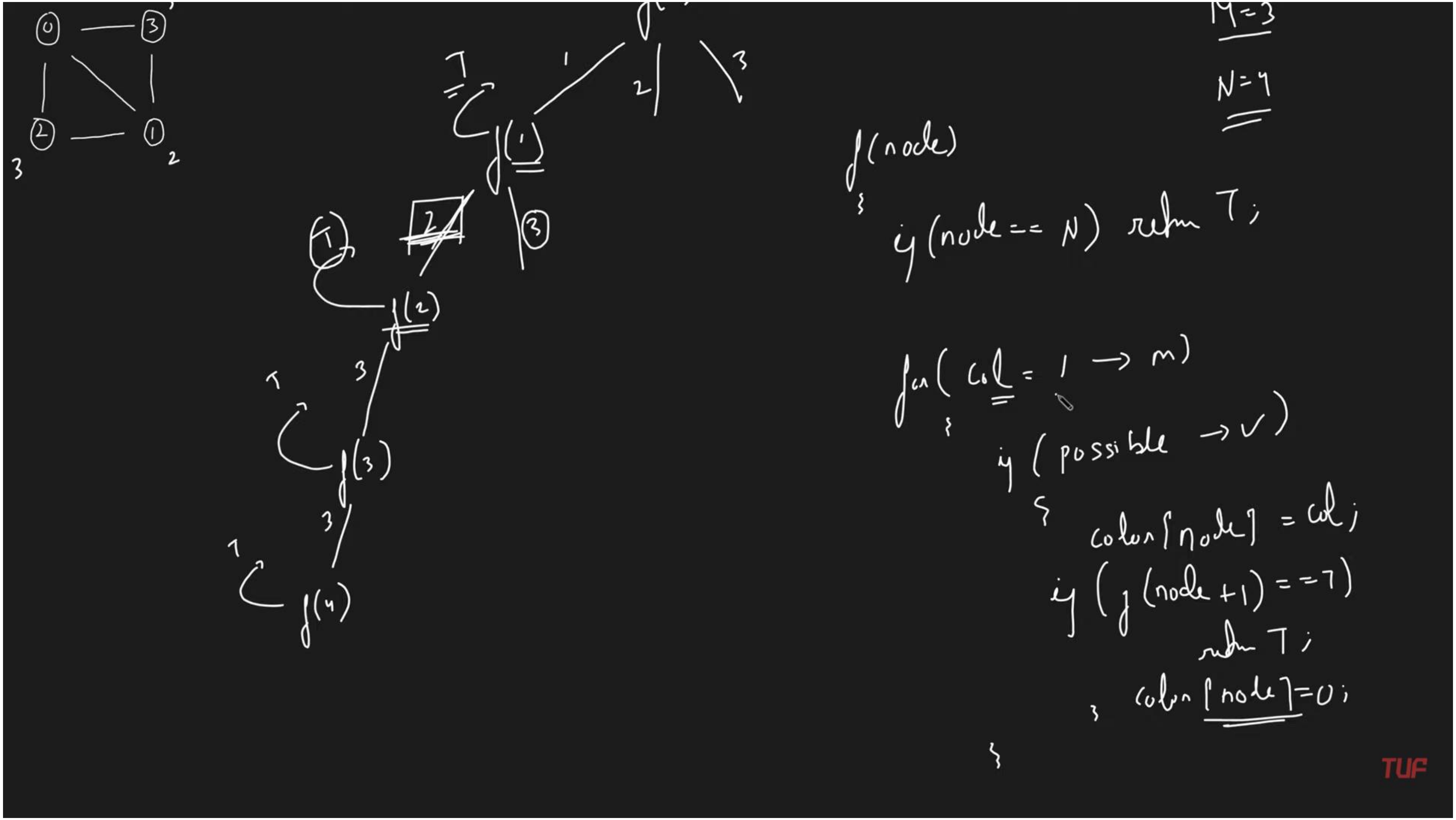


M = 2



M=3

N=4

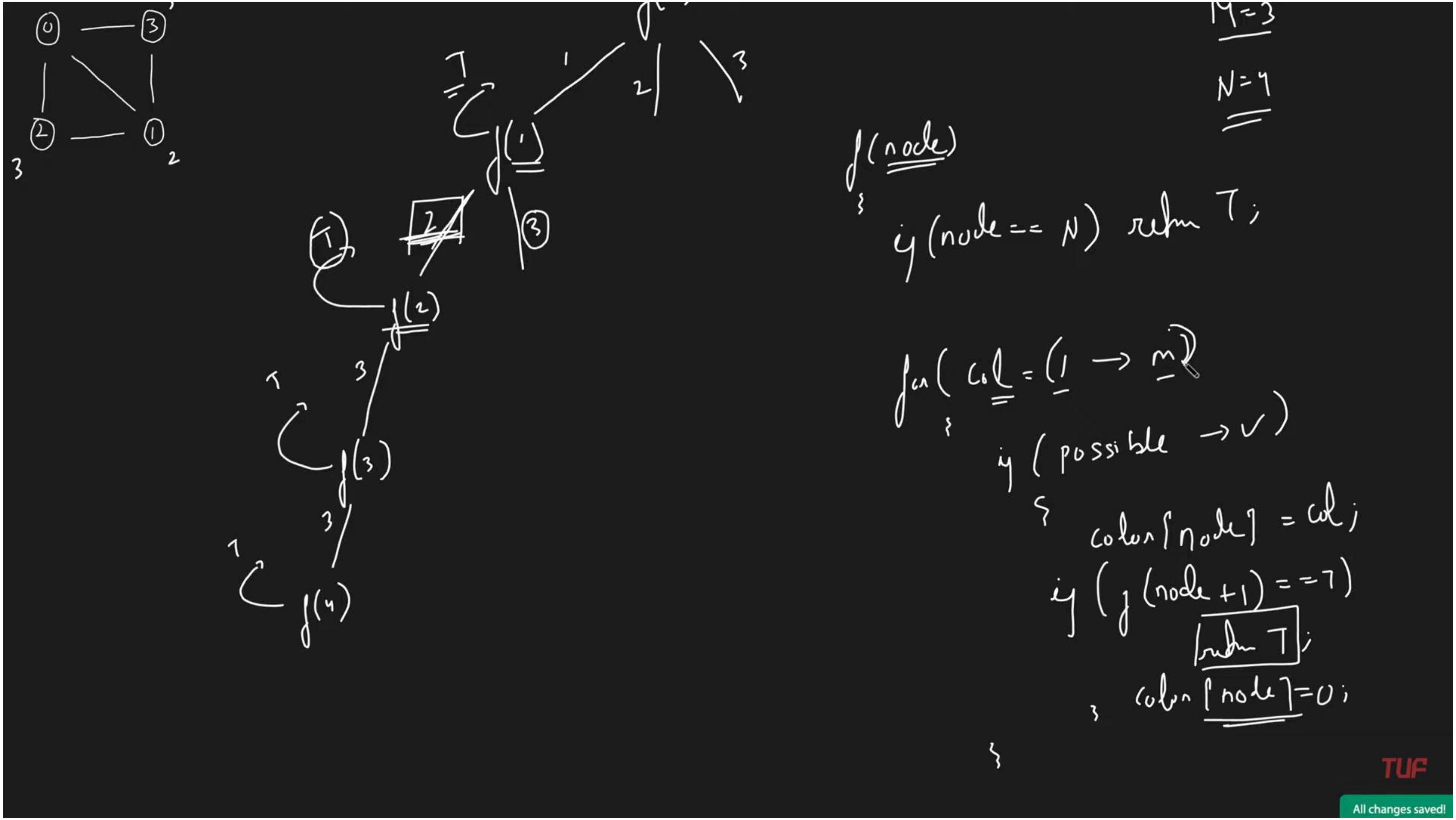


M=3

N=4

```
f(node)
{
   if (node == N) return T;

   for( col = 1 -> m)
   {
      if ( possible -> v)
      {
         color[node] = col;
         if (f(node+1) == T)
            return T;
         color[node]=0;
      }
   }
}
```

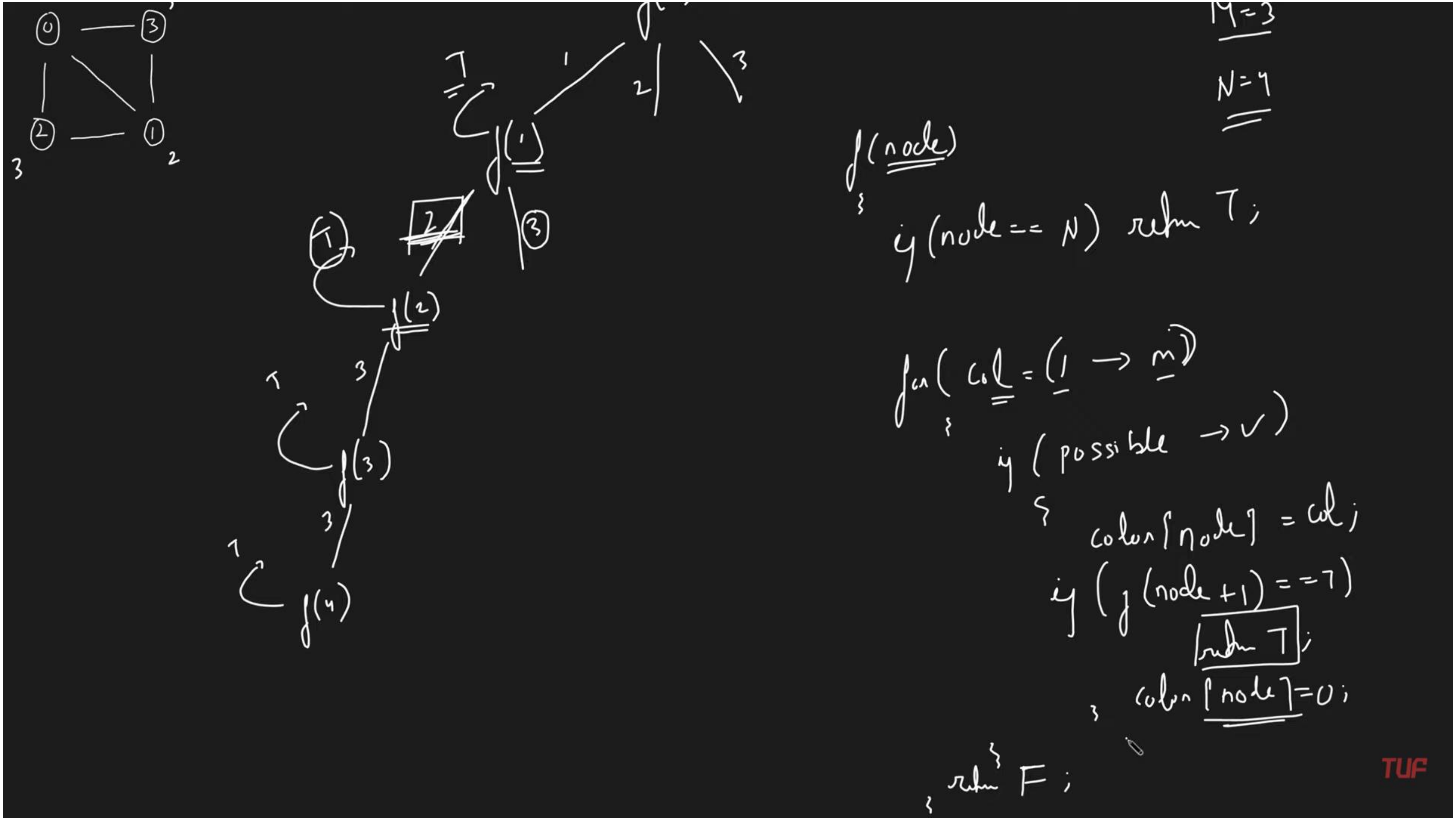


M=3

N=4

```
f(node)
{
    if (node == N) return T;

    for (col = (1 -> m)
    {
        if (possible -> v)
        {
            color[node] = col;
            if (f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
```

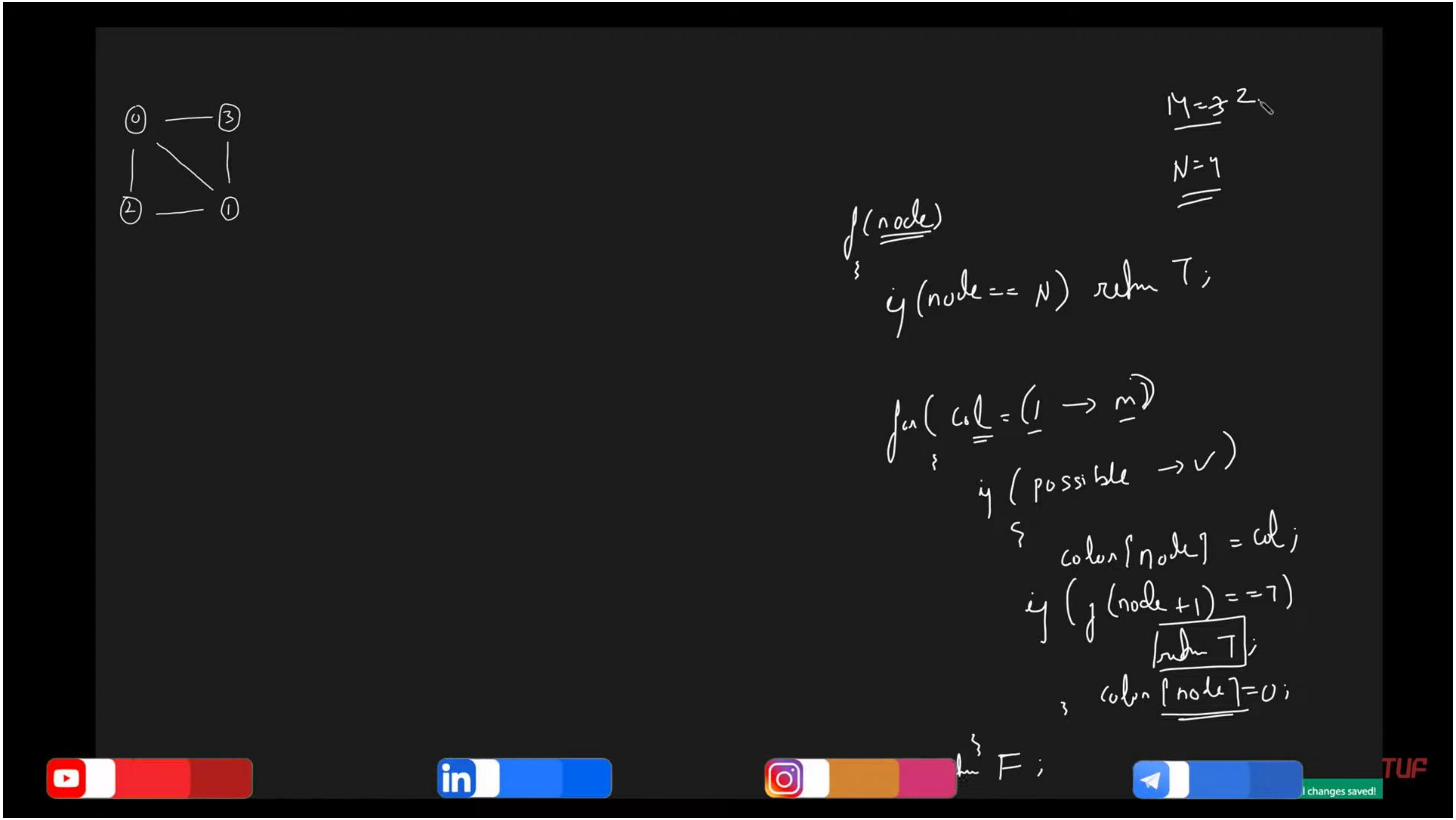


M=3

N=4

```
f(node)
{
    if(node == N) return T;

    for( col = (1 -> m))
    {
        if ( possible -> v)
        {
            color[node] = col;
            if ( f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
    return F;
}
```

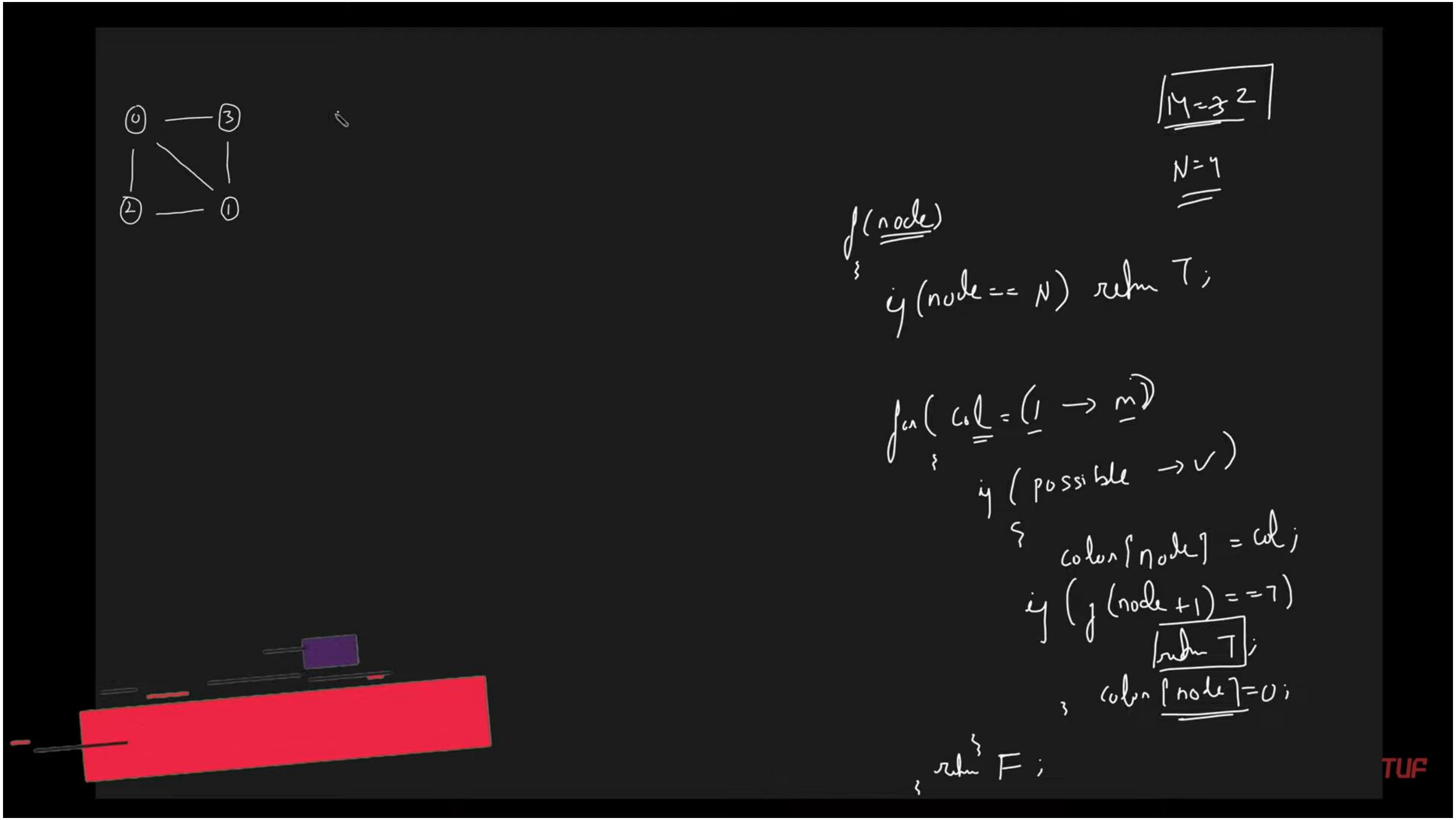


$M = \cancel{3}\ 2$

$N = 4$

```
f(node)
{
  if (node == N) return T;

  for (col = (1 -> m))
  {
    if (possible -> ✓)
    {
      color[node] = col;
      if (f(node+1) == T)
        return T;
    }
    color[node] = 0;
  }
  return F;
```

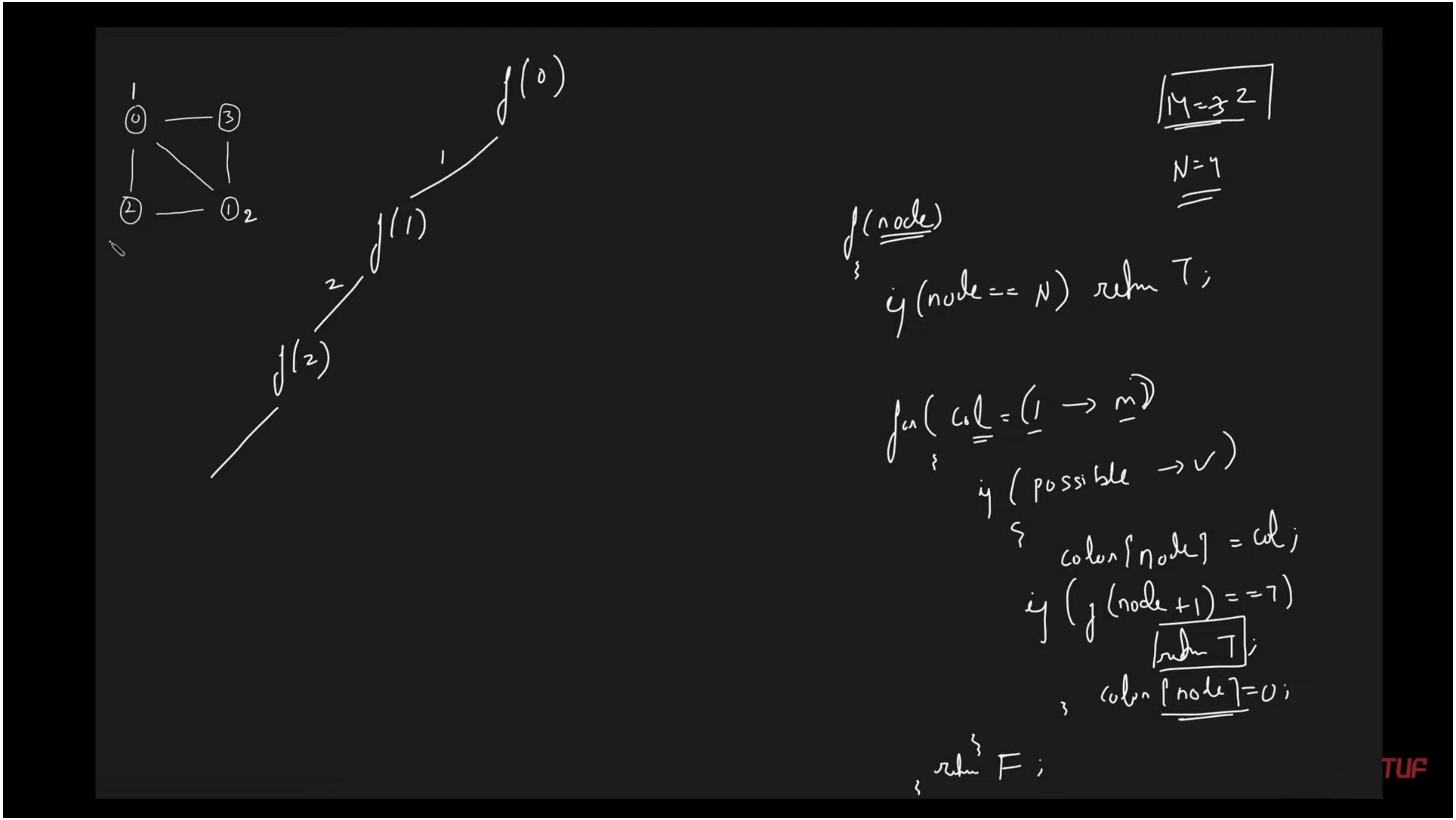


M = ~~3~~ 2

N = 4

```
f(node)
{
    if (node == N) return T;

    for (col = (1 -> m))
    {
        if (possible -> v)
        {
            color[node] = col;
            if (f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
    return F;
}
```

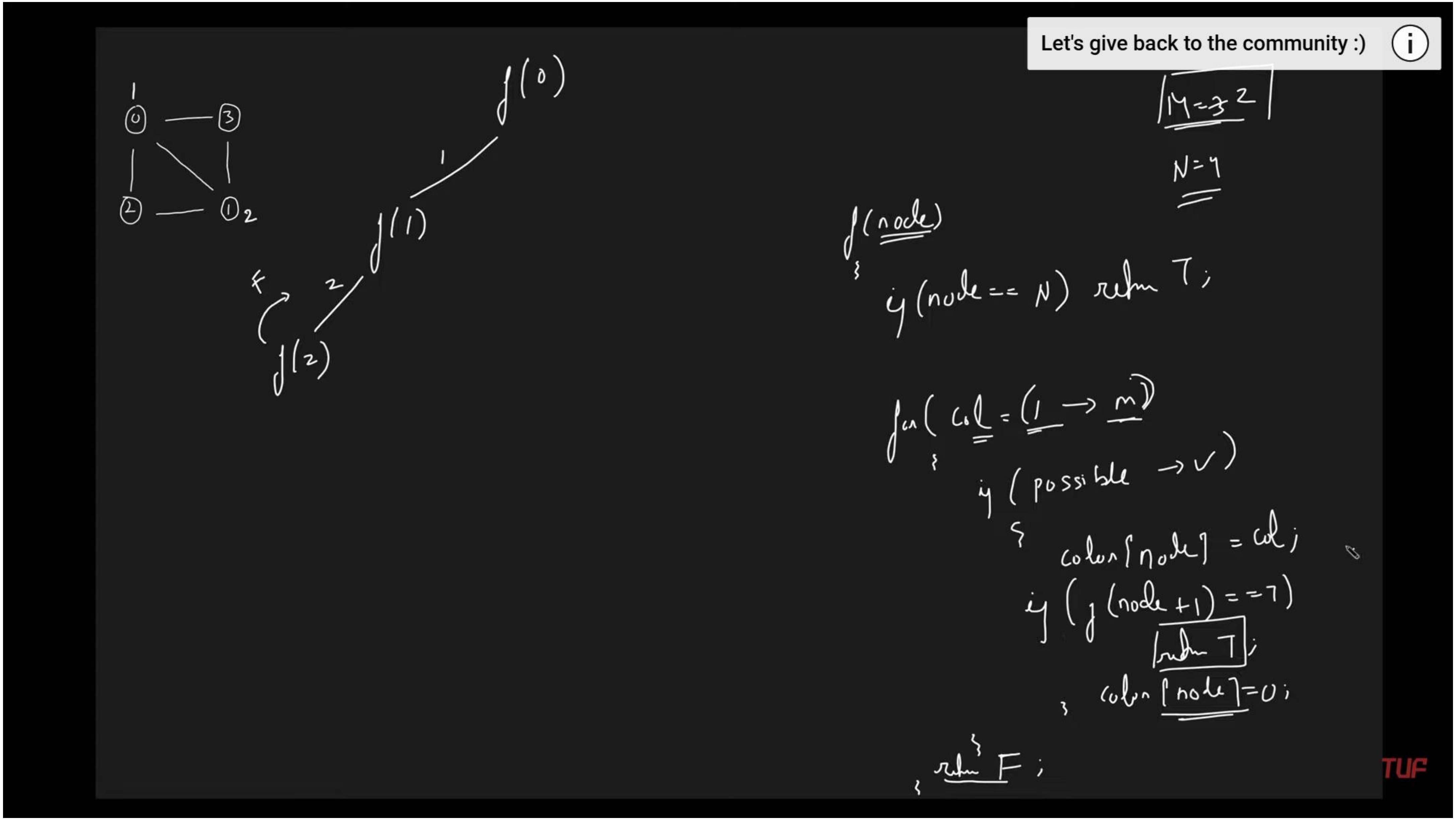


M = ~~3~~ 2

N = 4

```
f(node)
{
    if (node == N) return T;

    for ( col = (1 -> m))
    {
        if ( possible -> v)
        {
            color[node] = col;
            if ( f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
    return F;
}
```



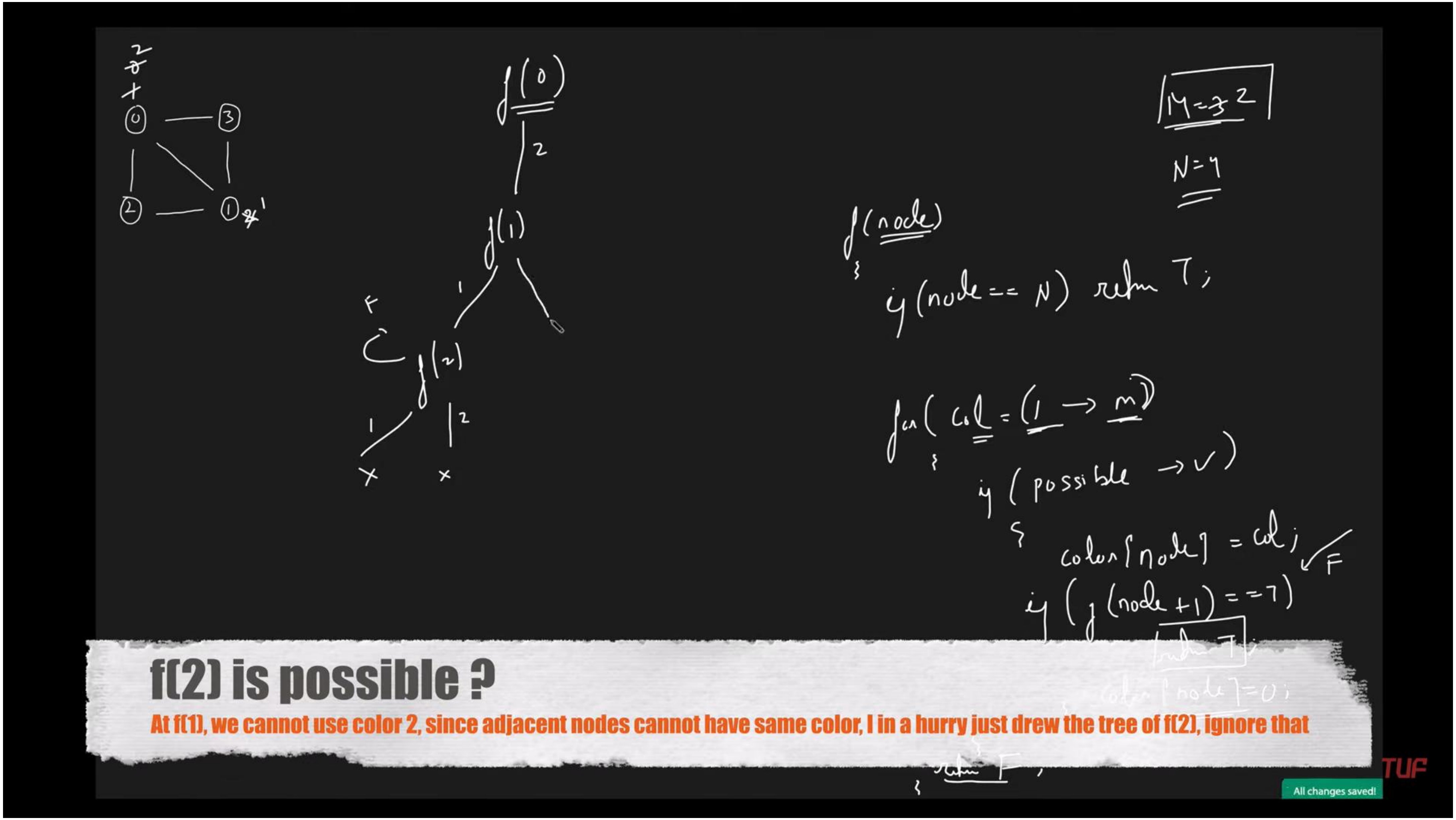
Let's give back to the community :)

M = ~~3~~ 2

N = 4

```
f(node)
{
    if (node == N) return T;

    for (col = (1 -> m))
    {
        if (possible -> v)
        {
            color[node] = col;
            if (f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
    return F;
}
```



M = ~~3~~ 2

N = 4

```
f(node)
{
    if (node == N) return T;

    for (col = (1 -> m))
    {
        if (possible -> v)
        {
            color[node] = col;
            if (f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
    return F;
}
```

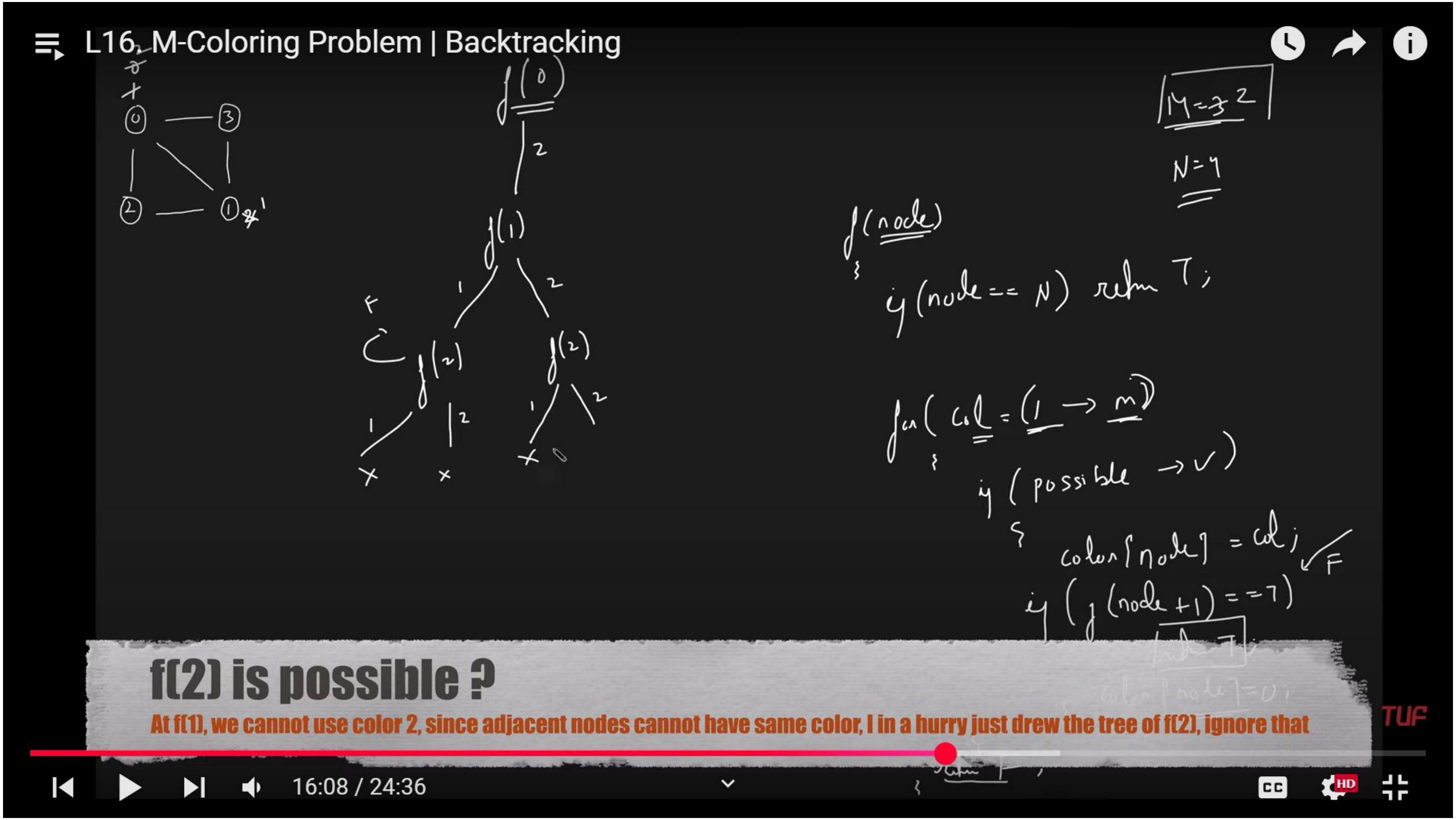
F

f(2) is possible ?

At f(1), we cannot use color 2, since adjacent nodes cannot have same color, I in a hurry just drew the tree of f(2), ignore that

TUF

All changes saved!



L16. M-Coloring Problem | Backtracking

M = ~~3~~ 2

N = 4

```
f(node)
{
    if (node == N) return T;

    for ( col = (1 -> m))
    {
        if ( possible -> v)
        {
            color[node] = col;
            if (f(node+1) == T)
                return T;
            color[node] = 0;
```

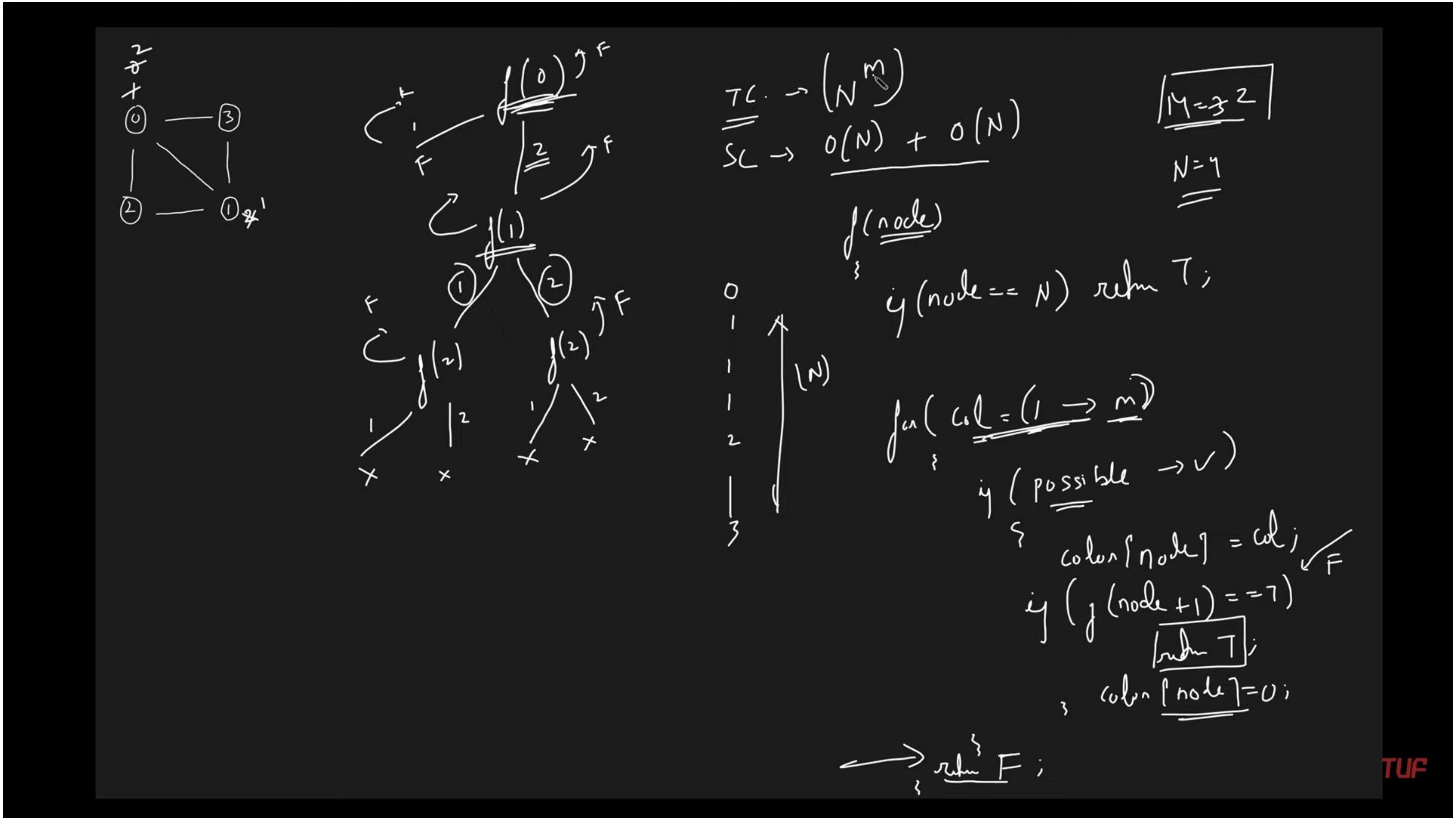
F

f(2) is possible ?

At f(1), we cannot use color 2, since adjacent nodes cannot have same color, I in a hurry just drew the tree of f(2), ignore that

TUF

16:08 / 24:36



TC → (N^m)

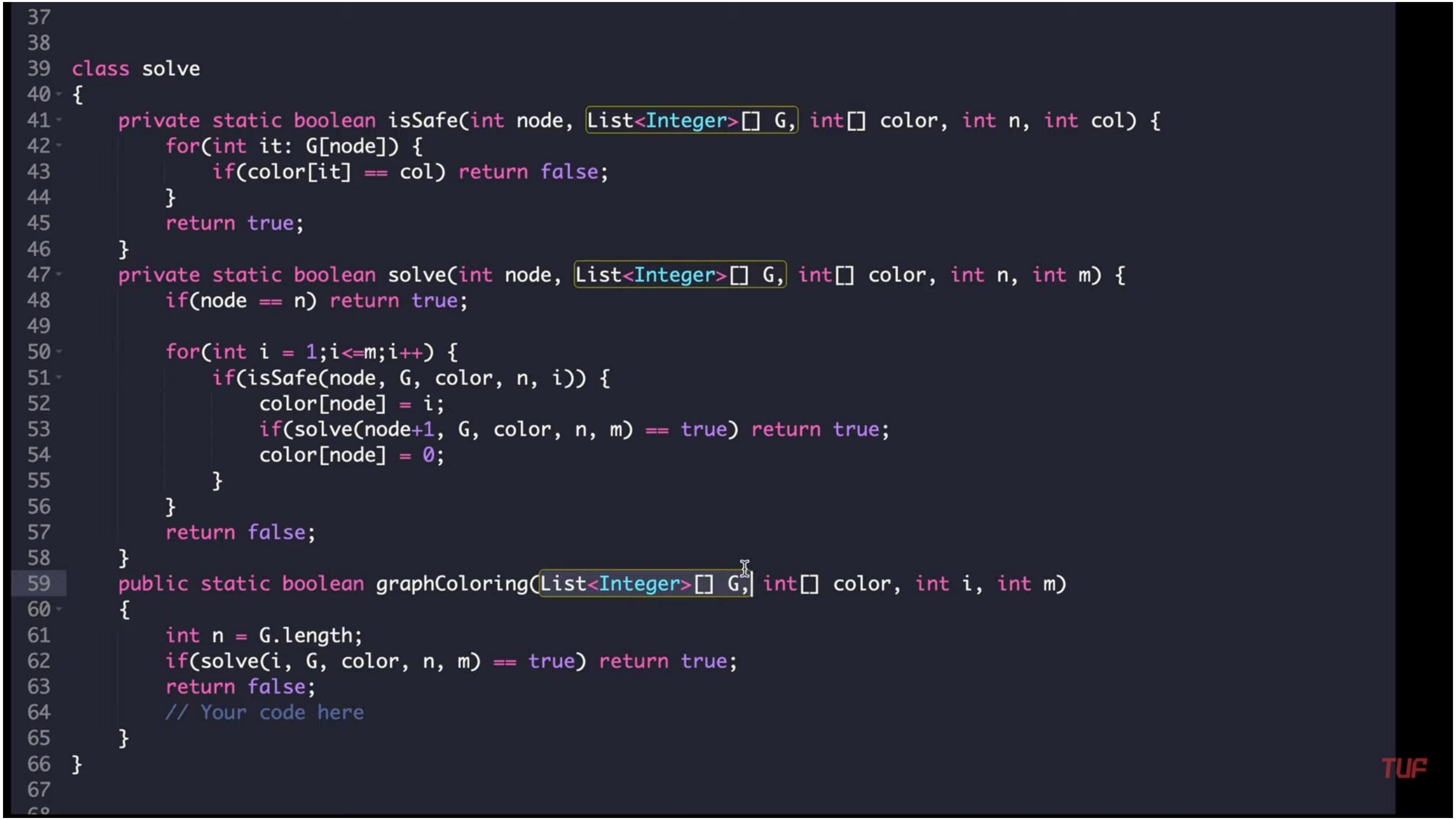
SC → O(N) + O(N)

M = ~~3~~ 2

N = 4

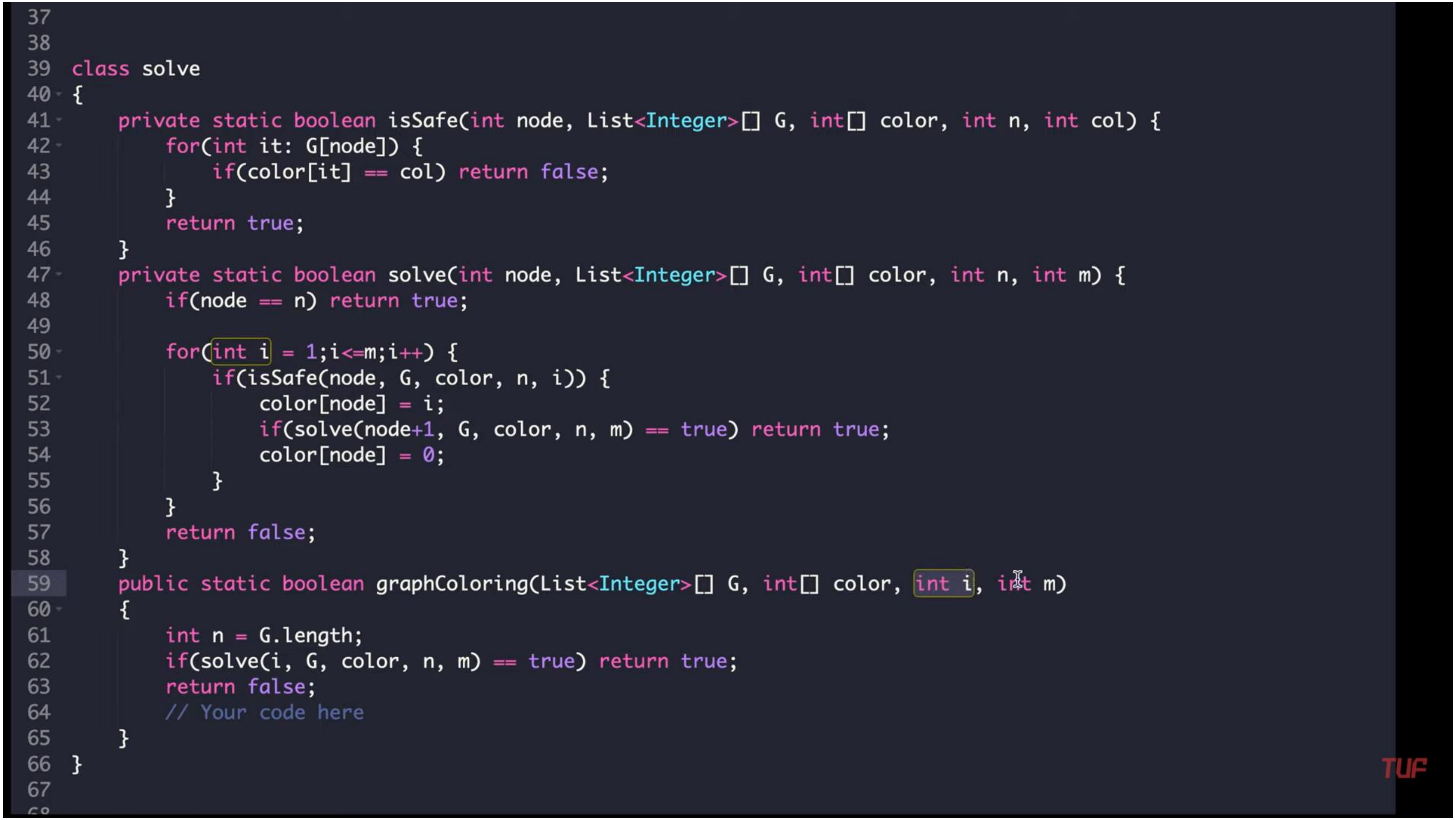
```
f(node)
{
    if (node == N) return T;

    for (col = (1 → m))
    {
        if (possible → v)
        {
            color[node] = col;
            if (f(node+1) == T)
                return T;
            color[node] = 0;
        }
    }
    return F;
}
```



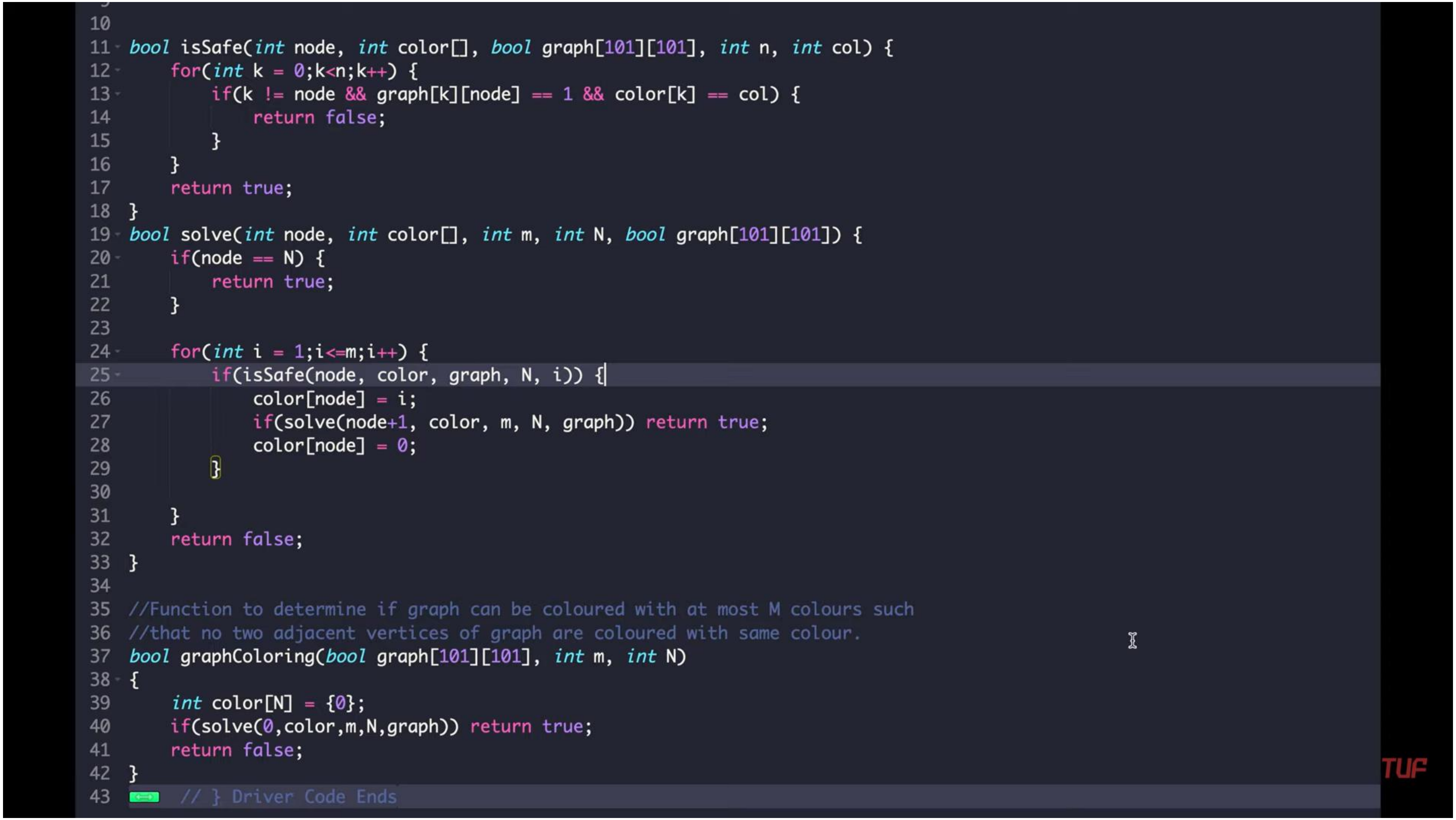
```java
class solve
{
    private static boolean isSafe(int node, List<Integer>[] G, int[] color, int n, int col) {
        for(int it: G[node]) {
            if(color[it] == col) return false;
        }
        return true;
    }
    private static boolean solve(int node, List<Integer>[] G, int[] color, int n, int m) {
        if(node == n) return true;

        for(int i = 1;i<=m;i++) {
            if(isSafe(node, G, color, n, i)) {
                color[node] = i;
                if(solve(node+1, G, color, n, m) == true) return true;
                color[node] = 0;
            }
        }
        return false;
    }
    public static boolean graphColoring(List<Integer>[] G, int[] color, int i, int m)
    {
        int n = G.length;
        if(solve(i, G, color, n, m) == true) return true;
        return false;
        // Your code here
    }
}
```



```
class solve
{
    private static boolean isSafe(int node, List<Integer>[] G, int[] color, int n, int col) {
        for(int it: G[node]) {
            if(color[it] == col) return false;
        }
        return true;
    }
    private static boolean solve(int node, List<Integer>[] G, int[] color, int n, int m) {
        if(node == n) return true;

        for(int i = 1;i<=m;i++) {
            if(isSafe(node, G, color, n, i)) {
                color[node] = i;
                if(solve(node+1, G, color, n, m) == true) return true;
                color[node] = 0;
            }
        }
        return false;
    }
    public static boolean graphColoring(List<Integer>[] G, int[] color, int i, int m)
    {
        int n = G.length;
        if(solve(i, G, color, n, m) == true) return true;
        return false;
        // Your code here
    }
}
```



```cpp

bool isSafe(int node, int color[], bool graph[101][101], int n, int col) {
    for(int k = 0;k<n;k++) {
        if(k != node && graph[k][node] == 1 && color[k] == col) {
            return false;
        }
    }
    return true;
}
bool solve(int node, int color[], int m, int N, bool graph[101][101]) {
    if(node == N) {
        return true;
    }

    for(int i = 1;i<=m;i++) {
        if(isSafe(node, color, graph, N, i)) {
            color[node] = i;
            if(solve(node+1, color, m, N, graph)) return true;
            color[node] = 0;
        }

    }
    return false;
}

//Function to determine if graph can be coloured with at most M colours such
//that no two adjacent vertices of graph are coloured with same colour.
bool graphColoring(bool graph[101][101], int m, int N)
{
    int color[N] = {0};
    if(solve(0,color,m,N,graph)) return true;
    return false;
}
 // } Driver Code Ends
```