Problem Set 6

CS 5800

Submitted by : Hardik Bishnoi

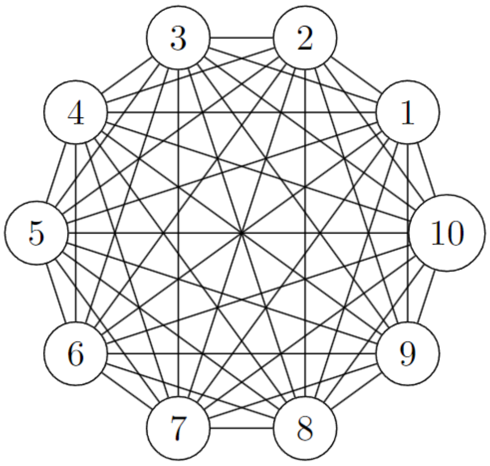
NUID : 002807991

Problem #1

(a) Draw the complete graph K_{10}, and determine the total number of edges in this graph. Briefly explain how you calculated this total.

Solution 1(a):

A graph K_{10} will be a graph with 10 vertices. We can represent K_{10} in the following graph:



This is a complete graph, since each vertex is connected with all the other 9 remaining vertices.

If we are to count the number of edges that are present in the graph, we can first count the number of edges that are connected with any arbitrary vertex. We know that in a complete graph any arbitrary vertex is connected to all the other $n-1$ vertices. For 10 total vertices, this amounts to $10-1=9$ edges per vertex.

If we then take the sum of all the edges connected to each individual point, we will have counted each edge twice. This means if the number of vertices is E:

$$(n-1)*n = 2E$$
$$\Rightarrow E = \frac{n(n-1)}{2}$$

Hence when $n = 10$:

$$E = \frac{10(10-1)}{2} = 45$$

Hence this graph has 45 edges.

(b) The complete graph K_n has exactly 120 edges. Determine the value of n. Clearly justify your answer.

Solution 1(b):

Since n is a big value, we need a formula to accurately calculate the number of vertices from the number of edges. Although we did derive the formula in the previous question 1(a), I will attempt to derive it here through a different method.

Suppose we have a complete graph with n vertices and E edges. An edge is made when there is a connection between any two vertices in a graph. Now, since a complete graph has any arbitrary edge connected with all the other edges, this means we can choose any two vertices out of n vertices to form an edge.

The number of unique combinations that can be made will be the total number of edges in the graph.

Hence,

$$E = \binom{n}{2}$$
$$E = \frac{n(n-1)}{2}$$

Since for this question, we're given $E = 120$, we can plug it into the above equation and get:

$$120 = \frac{n^2 - n}{2}$$
$$240 = n^2 - n$$
$$\Rightarrow n^2 - n - 240 = 0$$

Solving for n in the quadratic equation we have :

$$(n-15)(n+16) = 0 \Rightarrow n = 15, n = -16$$

Through the solution to the above equation, we have 15 and -16 as our roots. Since n cannot be negative, it must be $n = 15$.

<u>Hence, the number of vertices in a graph with 120 edges must be 15.</u>

(c) Sometime in 2021 (or 2022), a group of CS5800 students meet at the Northeastern campus for the first time, to have a post-Covid celebration party. Each pair of students shakes hands. Unfortunately, Paul walks in late. As a result, Paul is only able to shake hands with some of the other students at the celebration party. If there are exactly 42 handshakes in total, determine the number of hands that Paul shook. Clearly and carefully justify your answer.

Solution 1(c):

Let us assume that each person in the party is a vertex in a graph, and hence the handshakes between two persons in the celebration party can be denoted as an edge between the two person vertexes.

So, we can divide the given problem into two constituent parts : (1) The number of handshakes that happen before Paul arrives, (2) the number of handshakes Paul makes with people.

For the first case, we know that each person at the party except Paul shakes each others' hands. Hence, any arbitrary person would have shook all the other persons' hands. This means an edge is created from any arbitrary vertex to all the other $n-1$ vertices. This satisfies the definition of a complete graph. Now, we know that for a fixed n, we can define the number of edges in the graph, $E = \frac{n(n-1)}{2}$. Here, E is the total number of of handshakes before Paul arrives. The value of E must be the one that is lower than and closest to the value of the total handshakes.

Let us look at values of E for different values of n:

$$\begin{gathered} n = 1 \Rightarrow E = 0 \\ n = 2 \Rightarrow E = 1 \\ n = 3 \Rightarrow E = 3 \\ n = 4 \Rightarrow E = 6 \\ \dots \\ n = 9 \Rightarrow E = 36 \\ n = 10 \Rightarrow E = 45 \end{gathered}$$

Since $E_9 < 42$ and $E_{10} > 42$, the value of E must be equal to the value of E_9.

This means 9 people made 36 handshakes before Paul arrived. After Paul arrived, he made $42 - 36 = 6$ handshakes. This means there are $n + 1 = 10$ people at the celebration party.

Problem #2

(a) In your own words, describe how Breadth-First Search (BFS) and Depth-First Search (DFS) work. Does one search algorithm always reach the destination faster than the other? Explain.

Solution 2(a):

Breadth-First Search and Depth-First search are algorithms which are used to traverse or explore a given graph.

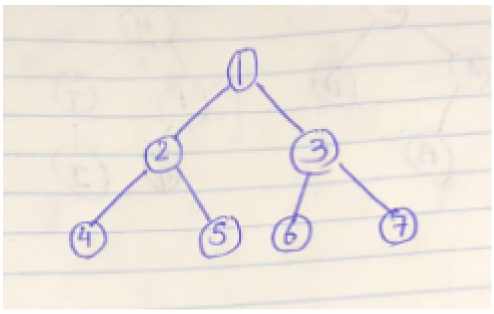
Breadth-First Search

Breadth-First Search or BFS is a graph traversal algorithm which explores a given graph level-by -level. BFS starts from a starting or source node and explores its neighbor nodes before moving to the next level of nodes.

The BFS algorithm uses the stack data structure to perform its operations in a FIFO order. The source node is enqueued into the queue first. If the queue is not empty, we pop the element which went in first and then check its neighbors. If there are any neighbors present we will enqueue them into the queue and mark them as visited. We repeat this till the queue is empty/the graph is completely traversed.

Now, I will explain how BFS works through a step by step demonstration.

Suppose we have the following graph:



Let us initialize a queue Q and an array V to store all the visited nodes :

$$V : []$$
$$Q : []$$

Where $\leftarrow$ show the direction of push and pop operations in the queue.

Now, first we push or enqueue the source node or 1 into the queue.

$$V : [1]$$
$$Q : [1]$$

This marks the source node as visited, and pop/dequeue 1.

Next, we explore any neighbors of 1 if they exist. Since 2 and 3 are neighbors of 1, we enqueue 2 and 3 into the queue, and mark them as visited.

$$V : [1, 2, 3]$$
$$Q : [2, 3]$$

Next, since the queue is not empty, we will perform the dequeue operation and dequeue 2. Now, we check for the immediate neighbors of 2, which are 4 and 5, and enqueue them into the queue. We will also mark them as visited.

$$V : [1, 2, 3, 4, 5]$$
$$Q : [3, 4, 5]$$

Next, since the queue is not empty, we will dequeue 3 since it is first in out of 3,4,5. We check for any neighbors that 3 has, and we find that it does have neighbors 6 and 7.

$$V : [1, 2, 3, 4, 5, 6, 7]$$
$$Q : [4, 5, 6, 7]$$

Next, since the queue is not empty, we will dequeue 4. We check for any neighbors that 4 has and do not find any.

$$V : [1, 2, 3, 4, 5, 6, 7]$$
$$Q : [5, 6, 7]$$

Similarly we will dequeue 5, 6 and 7 since they do not have any neighbors. We've now traversed the entire graph using BFS.

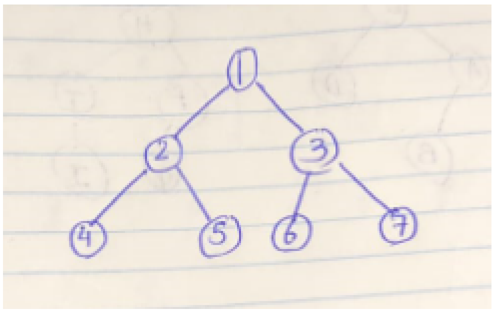
Let us now move on to Depth-First Search.

Depth-First Search:

Depth-First Search is a graph traversal algorithm which explores a graph as far as possible along each branch before backtracking. The DFS algorithm uses a stack based implementation where nodes are sent in the stack in a LIFO order. In DFS, the source node is pushed into the stack first and marked as visited. Then, the algorithm takes the vertex at the top of the stack and checks for any adjacent vertices that are not visited. If such vertices exist, we push them to the top of the stack. If there are no new vertices to visit, we pop the topmost vertex from the stack and repeat the search for any unvisited vertex. We keep doing this till all the vertices are popped out of the stack.

Now, I will explain DFS using an example.

Suppose we have the following graph:



Let us initialize a stack S.

$$S = []$$

In the stack, the leftmost element is at the bottom and the rightmost is at the top.

First, we will input the source node into the stack:

$$S = [1]$$

Next, we will scan for any immediate neighbors that the stack has. We first find 2, hence we will push it to the stack :

$$S = [1, 2]$$

Next, we look for 2's immediate unexplored neighbors. We find 4, and push it to the stack.

$$S = [1, 2, 4]$$

However, now we do not have any immediate neighbors since 4 is a leaf node. Hence, we backtrack by popping 4 from the stack.

$$S = [1, 2]$$

We now search again for any unexplored neighbors and find 5. Hence we push it to the stack.

$$S = [1, 2, 5]$$

Again, since 5 is a leaf node, it does not have any children, hence we will pop 5 from the stack.

$$S = [1, 2]$$

Next, we do not have any unexplored neighbors left for 2. Hence we pop 2.

$$S = [1]$$

Now we input the other unexplored neighbor of 1, which is 3 into the stack.

$$S = [1, 3]$$

We then search for an unexplored neighbor for 3 and come across 6. We push it to the stack.

$$S = [1, 3, 6]$$

Since 6 is a leaf node, we will pop it from the stack.

$$S = [1, 3]$$

After this, we will check for 3's unexplored neighbors once again and find 7, hence pushing it to the top of the stack.

$$S = [1, 3, 7]$$

Since 7 is a leaf node, we will pop it from the stack.

$$S = [1, 3]$$

Since 3 does not have any unexplored neighbors, we will pop it from the stack.

$$S = [1]$$

Finally, since 1 does not have any unexplored neighbors, we will also pop the source from the stack.

$$S = []$$

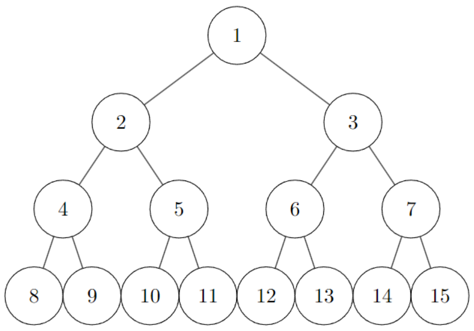
Since the stack is now empty, we have successfully traversed the tree with DFS.

BFS is **not necessarily faster** than DFS (or vice versa) to reach a given point. The question of which algorithm is faster depends on the location of the node. Suppose we have a tree with a very large height, DFS will be slower than BFS to find a node located at a shallow depth in the last branch. This is because DFS has to backtrack after traversing the entire branch to go to the next branch but BFS will consider the entire level of the tree altogether. However, if the node was located at the first branch of the tree at the bottom, the DFS algorithm will be faster since it directly traverses the branch in a depth-first manner, but BFS will have to search each level for the target node.

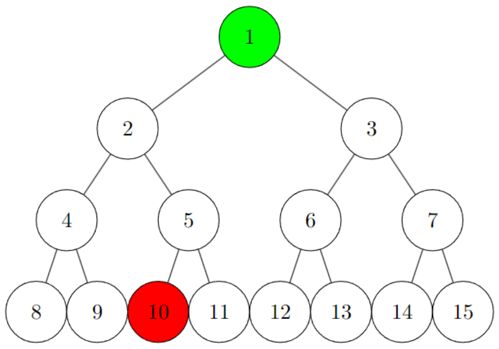
(b) Suppose we want to determine a path from vertex 1 (start vertex) to vertex 10 (end vertex). Using BFS, determine the order in which the vertices will be visited. Using DFS, determine the order in which the vertices will be visited. Briefly explain your answers.

Solution 2(b):

We have the following tree given to us with the target node



Our source node is 1 and our destination node is 10.



BFS

Let us first initialize a queue Q for the BFS. We will also initialize a array V to keep track of all the visited nodes while we perform BFS.

$$V = []$$
$$Q = []$$

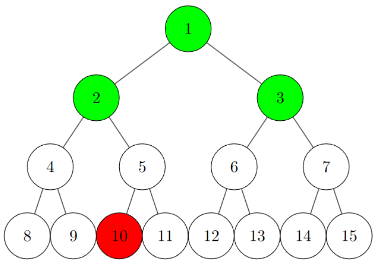
Next, we will input the source node into the queue and mark it as visited:

$$V = [1]$$
$$Q = [1]$$

Next, since the queue is not empty, we will dequeue 1 from the queue and search for the neighbors of queue and enqueue them if they exist. These are 2 and 3. We mark 2 and 3 as visited as well:

$$V = [1, 2, 3]$$
$$Q = [2, 3]$$

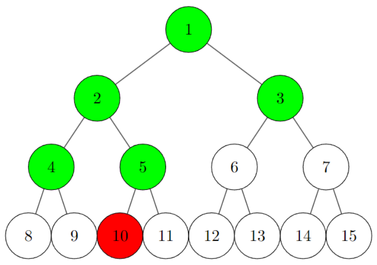
Here is how the tree will look like :



Again, since the queue is not empty we will pop 2 out of the queue and search for its neighbors which are 4 and 5 and then enqueue them into the queue. We will hence mark 4 and 5 as visited.

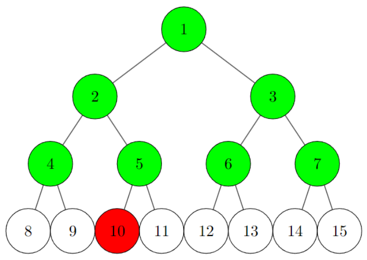
$$V = [1, 2, 3, 4, 5]$$
$$Q = [3, 4, 5]$$

Here is how the tree will look like :



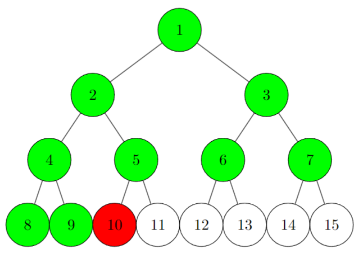
Since the queue is not empty we will dequeue 3 and search for its neighbors. We find 6 and 7 and enqueue it into the queue and mark them as visited.

$$V = [1, 2, 3, 4, 5, 6, 7]$$
$$Q = [4, 5, 6, 7]$$



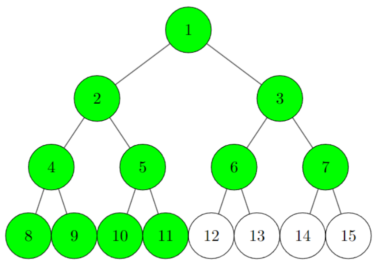
Next, since the queue is not empty, we will dequeue 4 and search for its neighbors. We find 8 and 9 and we enqueue them into the queue and mark them as visited.

$$V = [1, 2, 3, 4, 5, 6, 7, 8, 9]$$
$$Q = [5, 6, 7, 8, 9]$$



The queue is still not empty hence we dequeue the next element in line which is 5. We then search for the neighbors of 5 which are 10 and 11.

$$V = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$$
$$Q = [6, 7, 8, 9, 10, 11]$$

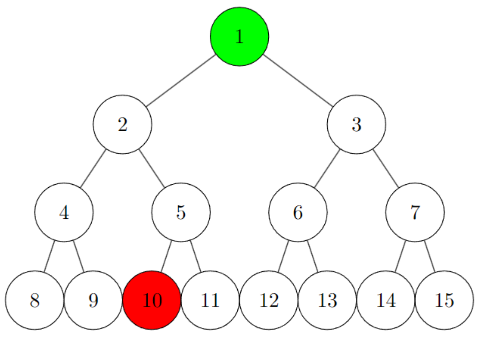


Since 10 is found, we have concluded the search.

The order of the visit is the order of the visited array V : [1,2,3,4,5,6,7,8,9,10,11].

Now, let us move to DFS:

DFS

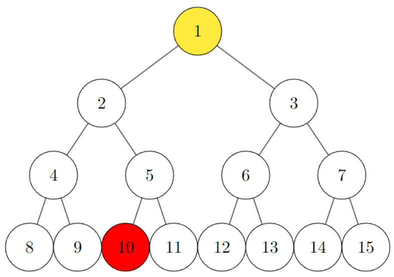


Let us first initialize a stack S and an array to keep track of the visited nodes V.

$$V = []$$
$$S = []$$

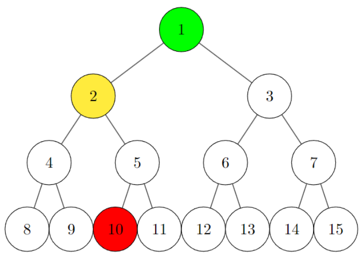
First we will input the source node, 1, into the stack. We will also mark it as visited.

$$V = [1]$$
$$S = [1]$$



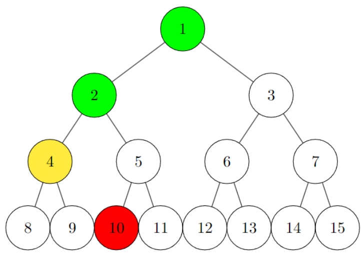
Now, since the stack is not empty, we will search for the immediate neighbors of 1. We find 2 and we put it in the stack and mark it as visited.

$$V = [1, 2]$$
$$S = [1, 2]$$



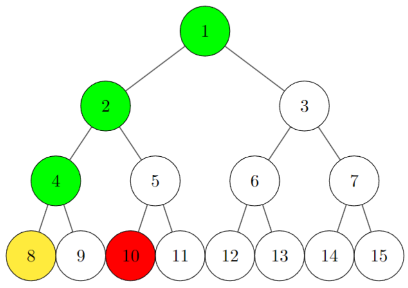
Next, since 2 is on top of the stack we will look at the immediate neighbors of 2. We find 4and put it at the top of the stack and mark it as visited.

$$V = [1, 2, 4]$$
$$S = [1, 2, 4]$$



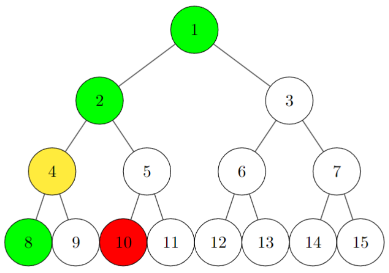
Next, we look for the immediate neighbors of 4. We find 8 and put it on the top of the stack and mark it as visited.

$$V = [1, 2, 4, 8]$$
$$S = [1, 2, 4, 8]$$



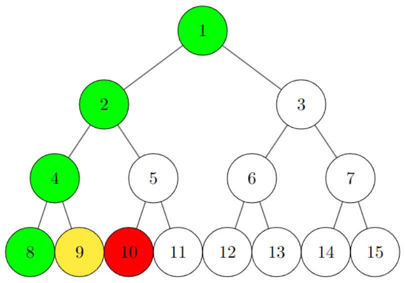
Since 8 is a leaf node it does not have any children. Hence we pop it and backtrack to 4.

$$V = [1, 2, 4, 8]$$
$$S = [1, 2, 4]$$



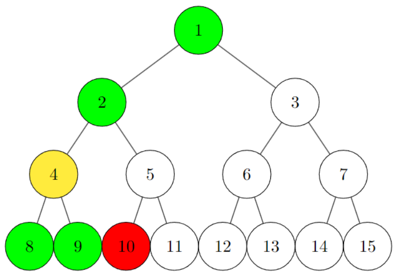
We find another unvisited neighbor 9 and we push it to the top of the stack and mark it as visited.

$$V = [1, 2, 4, 8, 9]$$
$$S = [1, 2, 4, 9]$$



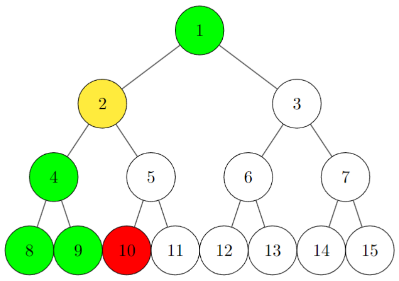
Again, since 9 is also a leaf node we cannot traverse further to a neighbor, hence we will backtrack by popping 9 out of the stack and back to 4.

$$V = [1, 2, 4, 8, 9]$$
$$S = [1, 2, 4]$$



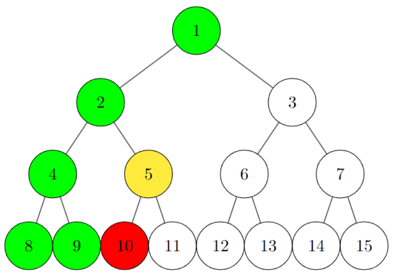
Since 4 does not have any other unexplored neighbors we will pop 4 and backtrack to 2.

$$V = [1, 2, 4, 8, 9]$$
$$S = [1, 2]$$



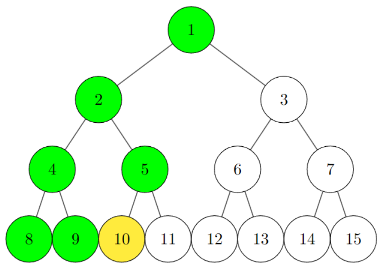
Now, we search for any unexplored neighbors that 2 may have. We find 5 and push it to the top of the queue and mark it as visited.

$$V = [1, 2, 4, 8, 9, 5]$$
$$S = [1, 2, 5]$$



Next, we will search for the neighbors of 5, we find 10 and push it to the stack and mark it as visited.

$$V = [1, 2, 4, 8, 9, 5, 10]$$
$$S = [1, 2, 5, 10]$$

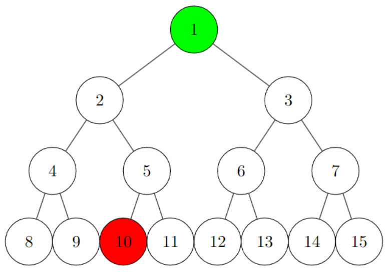


We have hence found 10, our destination node. Hence the search has concluded.

The order of visits is : [1,2,4,8,9,5,10] using DFS.

(c) Suppose that we extend this binary tree to infinitely many levels, so that each vertex k has two children: $2k$ (Left) and $2k+1$ (Right). The path from vertex 1 to vertex 10 can be described by a sequence of Left and Right moves, namely Left, Right, Left. Consider the path from vertex 1 to vertex 2021. Determine the sequence of Left and Right moves for this path. Clearly justify your answer.

Solution 2(c):



From the given example of the tree we know that that starting from the source node 1, the destination node 10 is in the order left, right left. By looking at the tree we can tell that all the even numbers will be a left child of the parent and the odd numbers will be the right child of the parent. Hence it becomes a matter of choosing the correct subtree at each level.

To find this, we can assume that the entire tree is in an array:

$$[1, 2, 3, 4, 5, 6, \ldots]$$

Next, we can find the parents of the child using the formula $\frac{index-1}{2}$. The index of 1 is 0, and the index of 2021 is 2020. We keep finding the parents of the parents till we reach the source node 1.

Hence, the parent of 2021 is $\frac{2020-1}{2} = 1009$, which is 1010

The parent of 1010 is $\frac{1009-1}{2} = 504$, which is the value 505

The parent of 505 is $\frac{504-1}{2} = 251$, which is the value 252

The parent of 252 is $\frac{251-1}{2} = 125$, which is the value 126

The parent of 126 is $\frac{125-1}{2} = 62$ which is the value 63

The parent of 63 is $\frac{62-1}{2} = 30$, which is the value 31

The parent of 31 is $\frac{30-1}{2} = 14$ which is the value 15

The parent of 15 is $\frac{14-1}{2} = 6$ which is the value 7

The parent of 7 is $\frac{6-1}{2} = 2$ which is the value 3

The parent of 3 is $\frac{2-1}{2} = 0$, which is the value 1 or the source node.

We can now traverse each node now that we know the order of the parents.

1, 3, 7, 15, 31, 63, 126, 252, 505, 1010, 2021

Since we know that the odd numbers mean right and the even numbers mean left,

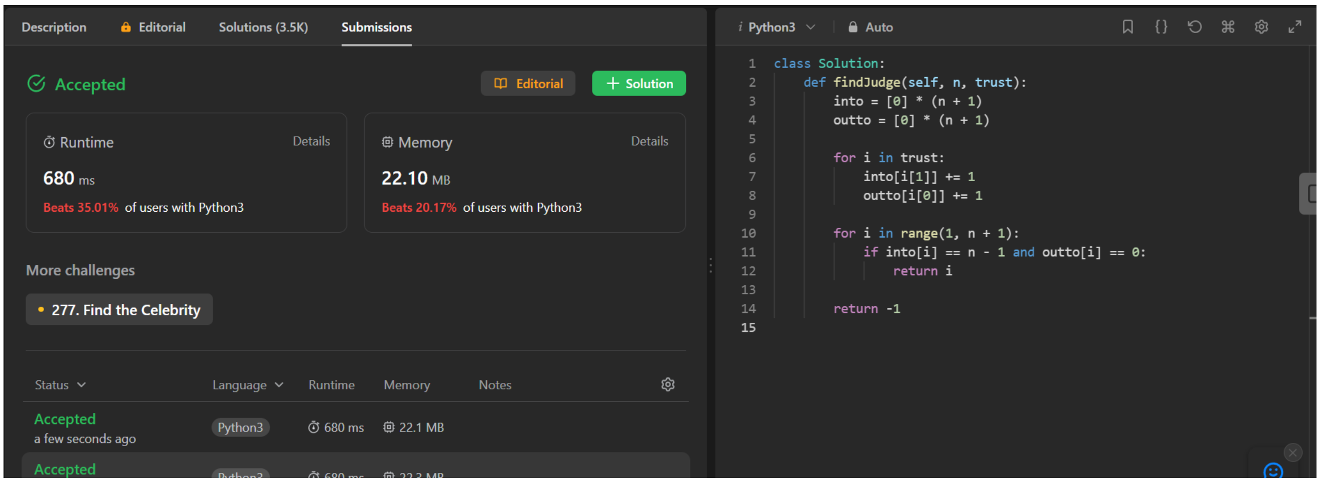
The order will be : 1, right, right, right, right, right, left, left, right, left, right.

Problem 3

Problem : Find the town judge

Problem number : Problem #997

Difficulty Level : Easy



Accepted

Runtime

680 ms

Beats 35.01% of users with Python3

Memory

22.10 MB

Beats 20.17% of users with Python3

More challenges

277. Find the Celebrity

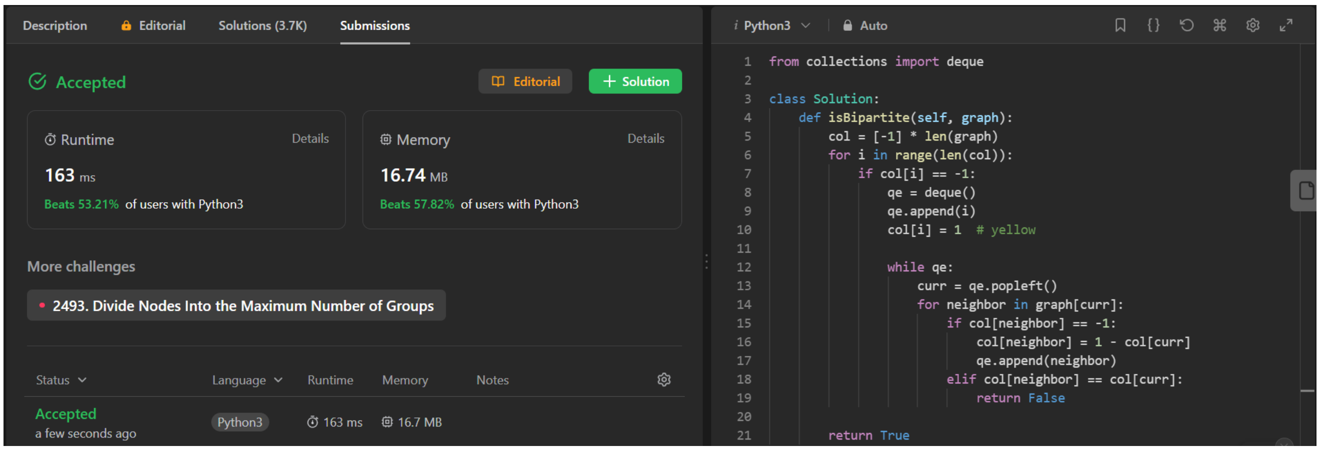
```
Status      Language    Runtime    Memory    Notes
Accepted                Python3    680 ms    22.1 MB
a few seconds ago
Accepted
```

```python
class Solution:
    def findJudge(self, n, trust):
        into = [0] * (n + 1)
        outto = [0] * (n + 1)

        for i in trust:
            into[i[1]] += 1
            outto[i[0]] += 1

        for i in range(1, n + 1):
            if into[i] == n - 1 and outto[i] == 0:
                return i

        return -1
```

Problem : Is Graph Bipartite?

Problem number : Problem #785

Difficulty Level : Medium



Accepted

Runtime

163 ms

Beats 53.21% of users with Python3

Memory

16.74 MB

Beats 57.82% of users with Python3

More challenges

2493. Divide Nodes Into the Maximum Number of Groups

```
Status      Language    Runtime    Memory    Notes
Accepted                Python3    163 ms    16.7 MB
a few seconds ago
```

```python
from collections import deque

class Solution:
    def isBipartite(self, graph):
        col = [-1] * len(graph)
        for i in range(len(col)):
            if col[i] == -1:
                qe = deque()
                qe.append(i)
                col[i] = 1  # yellow

                while qe:
                    curr = qe.popleft()
                    for neighbor in graph[curr]:
                        if col[neighbor] == -1:
                            col[neighbor] = 1 - col[curr]
                            qe.append(neighbor)
                        elif col[neighbor] == col[curr]:
                            return False

        return True
```

I collaborated with **Hakshay Sundar** on the last problem.