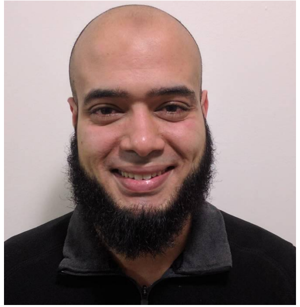
Algorithms

BFS Homework 4

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #1: LeetCode 1129 - Shortest Path with Alternating Colors

Consider a directed graph, with nodes labelled `0, 1, ..., n-1`. In this graph, each edge is either red or blue, and there could be self-edges or parallel edges.

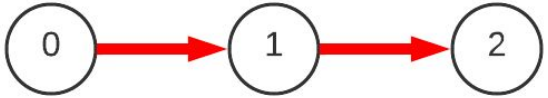
Each `[i, j]` in `red_edges` denotes a red directed edge from node `i` to node `j`. Similarly, each `[i, j]` in `blue_edges` denotes a blue directed edge from node `i` to node `j`.

Return an array `answer` of length `n`, where each `answer[X]` is the length of the shortest path from node `0` to node `X` such that the edge colors alternate along the path (or `-1` if such a path doesn't exist).

- C++: vector<int> shortestAlternatingPaths(int n, vector<vector<int>>& red_edges, vector<vector<int>>& blue_edges)
- Java: public int[] shortestAlternatingPaths(int n, int[][] red_edges, int[][] blue_edges)
- Python: def shortestAlternatingPaths(self, n, red_edges, blue_edges)
- Javascript: var shortestAlternatingPaths = function(n, red_edges, blue_edges)

Example 1:

```
Input: n = 3, red_edges = [[0,1],[1,2]], blue_edges = [
Output: [0,1,-1]
```

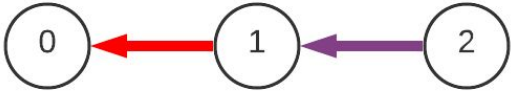


Example 2:

```
Input: n = 3, red_edges = [[0,1]], blue_edges = [[2,1]]
Output: [0,1,-1]
```

Example 3:

```
Input: n = 3, red_edges = [[1,0]], blue_edges = [[2,1]]
Output: [0,-1,-1]
```

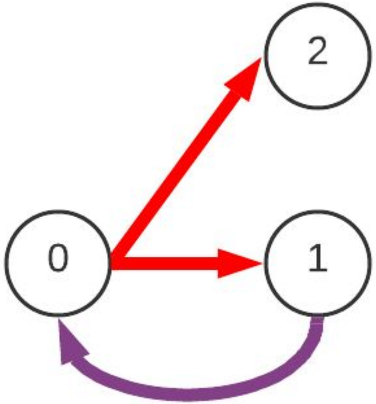


Example 4:

```
Input: n = 3, red_edges = [[0,1]], blue_edges = [[1,2]]
Output: [0,1,2]
```

Example 5:

```
Input: n = 3, red_edges = [[0,1],[0,2]], blue_edges = [[1,0]]
Output: [0,1,1]
```



Problem #2: LeetCode 365 - Water and Jug Problem

- You are given **two jugs** with capacities *jug1Capacity* and *jug2Capacity* liters.
- We can do any of these 6 operations, **as many as** we want:
    - **Fill** any of the jugs with water.
    - **Empty** any of the jugs.
    - **Pour** water from one jug into another till the other jug is completely full, or the first jug itself is empty.

  - Given targetCapacity liters, can you apply the operations to reach:
    - **Jug1_cur_water + Jug2_cur_water == targetCapacity**
- 1 <= jug1Capacity, jug2Capacity, targetCapacity <= $\mathbf{10^6}$
- C++: bool canMeasureWater(int jug1Capacity, int jug2Capacity, int targetCapacity)
- Java: public boolean canMeasureWater(int jug1Capacity, int jug2Capacity, int targetCapacity)
- Python: def canMeasureWater(self, jug1Capacity: int, jug2Capacity: int, targetCapacity: int) -> bool:
- Javascript: var canMeasureWater = function(jug1Capacity, jug2Capacity, targetCapacity)

Examples

- jug1Capacity = 1, jug2Capacity = 2, targetCapacity = 3
  - True ⇒ Clearly adding them 1+2 ⇒ 3
- jug1Capacity = 10, jug2Capacity = 20, targetCapacity = 100
  - False ⇒ their max is 10+20
- jug1Capacity = 3, jug2Capacity = 5, targetCapacity = 6
  - Initially ⇒ (0, 0) ⇒ fill2 ⇒ (0, 5) pour 2 in 1 ⇒ (3, 2) ⇒ empty 2 ⇒ (3, 0) ⇒ pour 1 to 2 ⇒ (0, 3) ⇒ fill1 ⇒ (3, 3) which sum to 6
- Bouns: Modify your code to **print the shortest operations sequence**

Note

- LeetCode has very tight time limit. 90% your BFS code will TLE
  - There is O(1) solution - out of our scope
- To avoid their hassle, I provide you with input/output file to test your code
  - Number of cases, then 3 values per line (jug1 jug2 target)

```
8
1 2 3
3 4 5
4 3 5
2 6 5
3 5 7
15 20 100
21 29 15
100 118 39
```

```
Co
<term
1
1
1
0
1
0
1
0
```

Problem #3: LeetCode 773 - Sliding Puzzle

On an `2 x 3` board, there are five tiles labeled from `1` to `5`, and an empty square represented by `0`. A **move** consists of choosing `0` and a 4-directionally adjacent number and swapping it.

The state of the board is solved if and only if the board is `[[1,2,3],[4,5,0]]`.

Given the puzzle board `board`, return *the least number of moves required so that the state of the board is solved*. If it is impossible for the state of the board to be solved, return `-1`.

- C++: int slidingPuzzle(vector<vector<int>>& board)
- Java: public int slidingPuzzle(int[][] board)
- Python: def slidingPuzzle(self, board: List[List[int]]) -> int
- Javascript: var slidingPuzzle = function(board)

Example 1:


```
Input: board = [[1,2,3],[4,0,5]]
Output: 1
Explanation: Swap the 0 and the 5 in one move.
```

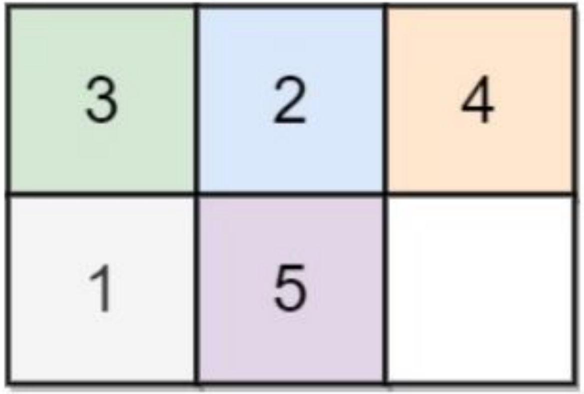
Example 2:


```
Input: board = [[1,2,3],[5,4,0]]
Output: -1
Explanation: No number of moves will make the board solved.
```

Example 3:


```
Input: board = [[4,1,2],[5,0,3]]
Output: 5
Explanation: 5 is the smallest number of moves that solves the board.
An example path:
After move 0: [[4,1,2],[5,0,3]]
After move 1: [[4,1,2],[0,5,3]]
After move 2: [[0,1,2],[4,5,3]]
After move 3: [[1,0,2],[4,5,3]]
After move 4: [[1,2,0],[4,5,3]]
After move 5: [[1,2,3],[4,5,0]]
```

Example 4:




```
Input: board = [[3,2,4],[1,5,0]]
Output: 14
```

Constraints:

- `board.length == 2`
- `board[i].length == 3`
- `0 <= board[i][j] <= 5`
- Each value `board[i][j]` is **unique**.

Problem #4: LeetCode 1298 - Maximum Candies You Can Get from Boxes

Given `n` boxes, each box is given in the format `[status, candies, keys, containedBoxes]` where:

- `status[i]` : an integer which is **1** if `box[i]` is open and **0** if `box[i]` is closed.
- `candies[i]` : an integer representing the number of candies in `box[i]` .
- `keys[i]` : an array contains the indices of the boxes you can open with the key in `box[i]` .
- `containedBoxes[i]` : an array contains the indices of the boxes found in `box[i]` .

You will start with some boxes given in `initialBoxes` array. You can take all the candies in any open box and you can use the keys in it to open new boxes and you also can use the boxes you find in it.

Return *the maximum number of candies* you can get following the rules above.

- C++: int maxCandies(vector<int> &hasKey, vector<int> &candies, vector<vector<int>> &keys, vector<vector<int>> &containedBoxes, vector<int> &initialBoxes)
- Python: def maxCandies(self, status: List[int], candies: List[int], keys: List[List[int]], containedBoxes: List[List[int]], initialBoxes: List[int]) -> int:

Example 1:

```
Input: status = [1,0,1,0], candies = [7,5,4,100], keys = [[],[],[1],[]], containedBoxes =
[[1,2],[3],[],[]], initialBoxes = [0]
Output: 16
Explanation: You will be initially given box 0. You will find 7 candies in it and boxes 1
and 2. Box 1 is closed and you don't have a key for it so you will open box 2. You will
find 4 candies and a key to box 1 in box 2.
In box 1, you will find 5 candies and box 3 but you will not find a key to box 3 so box 3
will remain closed.
Total number of candies collected = 7 + 4 + 5 = 16 candy.
```

Example 2:

```
Input: status = [1,0,0,0,0,0], candies = [1,1,1,1,1,1], keys = [[1,2,3,4,5],[],[],[],[],
[]], containedBoxes = [[1,2,3,4,5],[],[],[],[],[]], initialBoxes = [0]
Output: 6
Explanation: You have initially box 0. Opening it you can find boxes 1,2,3,4 and 5 and
their keys. The total number of candies will be 6.
```

Example 3:

```
Input: status = [1,1,1], candies = [100,1,100], keys = [[],[0,2],[]], containedBoxes =
[[],[],[]], initialBoxes = [1]
Output: 1
```

Example 4:

```
Input: status = [1], candies = [100], keys = [[]], containedBoxes = [[]], initialBoxes =
[]
Output: 0
```

Example 5:

```
Input: status = [1,1,1], candies = [2,3,2], keys = [[],[],[]], containedBoxes = [[],[],
[]], initialBoxes = [2,1,0]
Output: 7
```

Constraints:

- `1 <= status.length <= 1000`
- `status.length == candies.length == keys.length == containedBoxes.length == n`
- `status[i]` is `0` or `1`.
- `1 <= candies[i] <= 1000`
- `0 <= keys[i].length <= status.length`
- `0 <= keys[i][j] < status.length`
- All values in `keys[i]` are unique.
- `0 <= containedBoxes[i].length <= status.length`
- `0 <= containedBoxes[i][j] < status.length`
- All values in `containedBoxes[i]` are unique.
- Each box is contained in one box at most.
- `0 <= initialBoxes.length <= status.length`
- `0 <= initialBoxes[i] < status.length`

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”