Algorithms

BFS Homework 2

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 1306 - Jump Game III

Given an array of non-negative integers `arr`, you are initially positioned at `start` index of the array. When you are at index `i`, you can jump to `i + arr[i]` or `i - arr[i]`, check if you can reach to **any** index with value 0.

Notice that you can not jump outside of the array at any time.

- C++: bool canReach(vector<int>& arr, int start)
- Java: public boolean canReach(int[] arr, int start)
- Python: def canReach(self, arr: List[int], start: int) -> bool
- Javascript: var canReach = function(arr, start)

Example 1:

```
Input: arr = [4,2,3,0,3,1,2], start = 5
Output: true
Explanation:
All possible ways to reach at index 3 with value 0 are:
index 5 -> index 4 -> index 1 -> index 3
index 5 -> index 6 -> index 4 -> index 1 -> index 3
```

Example 2:

```
Input: arr = [4,2,3,0,3,1,2], start = 0
Output: true
Explanation:
One possible way to reach at index 3 with value 0 is:
index 0 -> index 4 -> index 1 -> index 3
```

Example 3:

```
Input: arr = [3,0,2,1,2], start = 2
Output: false
Explanation: There is no way to reach at index 1 with value 0.
```

arr[3] is zero
So this is the only valid end goal

Start from idx 5 and go to idx 3 using given operations

Problem #2: LeetCode 2059 - Minimum Operations to Convert Number

2059. Minimum Operations to Convert Number

Medium 234 15 Add to List Share

You are given a **0-indexed** integer array `nums` containing **distinct** numbers, an integer `start`, and an integer `goal`. There is an integer `x` that is initially set to `start`, and you want to perform operations on `x` such that it is converted to `goal`. You can perform the following operation repeatedly on the number `x`:

If `0 <= x <= 1000`, then for any index `i` in the array (`0 <= i < nums.length`), you can set `x` to any of the following:

- `x + nums[i]`
- `x - nums[i]`
- `x ^ nums[i]` (bitwise-XOR)

Note that you can use each `nums[i]` any number of times in any order. Operations that set `x` to be out of the range `0 <= x <= 1000` are valid, but no more operations can be done afterward.

Return *the* ***minimum*** *number of operations needed to convert* `x = start` *into* `goal`, *and* `-1` *if it is not possible*.

Problem #2: LeetCode 2059 - Minimum Operations to Convert Number

- C++: int minimumOperations(vector<int> &nums, int start, int goal)
- Java: public int minimumOperations(int[] nums, int start, int goal)
- Python: def minimumOperations(self, nums: List[int], start: int, goal: int) -> int
- JavaScript: var minimumOperations = function(nums, start, goal)

Constraints:

- `1 <= nums.length <= 1000`
- `-10^9 <= nums[i], goal <= 10^9`
- `0 <= start <= 1000`
- `start != goal`
- All the integers in `nums` are distinct.

Example 1:

```
Input: nums = [1,3], start = 6, goal = 4
Output: 2
Explanation:
We can go from 6 → 7 → 4 with the following 2 operations.
- 6 ^ 1 = 7
- 7 ^ 3 = 4
```

Example 2:

```
Input: nums = [2,4,12], start = 2, goal = 12
Output: 2
Explanation:
We can go from 2 → 14 → 12 with the following 2 operations.
- 2 + 12 = 14
- 14 - 2 = 12
```

Example 3:

```
Input: nums = [3,5,7], start = 0, goal = -4
Output: 2
Explanation:
We can go from 0 → 3 → -4 with the following 2 operations.
- 0 + 3 = 3
- 3 - 7 = -4
Note that the last operation sets x out of the range 0 <= x <= 1000, which is valid.
```

Example 4:

```
Input: nums = [2,8,16], start = 0, goal = 1
Output: -1
Explanation:
There is no way to convert 0 into 1.
```

Example 5:

```
Input: nums = [1], start = 0, goal = 3
Output: 3
Explanation:
We can go from 0 → 1 → 2 → 3 with the following 3 operations.
- 0 + 1 = 1
- 1 + 1 = 2
- 2 + 1 = 3
```

Problem #3: LeetCode 752 - Open the Lock

You have a lock in front of you with 4 circular wheels. Each wheel has 10 slots: `'0'`, `'1'`, `'2'`, `'3'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`. The wheels can rotate freely and wrap around: for example we can turn `'9'` to be `'0'`, or `'0'` to be `'9'`. Each move consists of turning one wheel one slot.

The lock initially starts at `'0000'`, a string representing the state of the 4 wheels.

You are given a list of `deadends` dead ends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a `target` representing the value of the wheels that will unlock the lock, return the minimum total number of turns required to open the lock, or -1 if it is impossible.

- C++: int openLock(vector<string> &deadends, string target)
- Java: public int openLock(String[] deadends, String target)
- Python: def openLock(self, deadends: List[str], target: str) -> int

Img src

Example 1:

```
Input: deadends = ["0201","0101","0102","1212","2002"], target = "0202"
Output: 6
Explanation:
A sequence of valid moves would be "0000" -> "1000" -> "1100" -> "1200" -> "1201" ->
"1202" -> "0202".
Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" -> "0202" would be
invalid,
because the wheels of the lock become stuck after the display becomes the dead end
"0102".
```

Example 2:

```
Input: deadends = ["8888"], target = "0009"
Output: 1
Explanation:
We can turn the last wheel in reverse to move from "0000" -> "0009".
```

Example 3:

```
Input: deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"], target =
"8888"
Output: -1
Explanation:
We can't reach the target without getting stuck.
```

Example 4:

```
Input: deadends = ["0000"], target = "8888"
Output: -1
```

- Notes:
  - start **may be** in the list deadends (then no solution)
  - start **may equal** target
  - target **will not** be in the list deadends.

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."