Leetcode 433.
Minimum Genetic Mutation

#Hash Table # String # Breadth-First Search

Problem Description

A gene string can be represented by an 8-character long string, with choices from 'A', 'C', 'G', and 'T'.

Suppose we need to investigate a mutation from a gene string startGene to a gene string endGene where one mutation is defined as one single character changed in the gene string.

For example, "AACCGGTT" --> "AACCGGTA" is one mutation.
There is also a gene bank bank that records all the valid gene mutations. A gene must be in bank to make it a valid gene string.

Given the two gene strings startGene and endGene and the gene bank bank, return the minimum number of mutations needed to mutate from startGene to endGene. If there is no such a mutation, return -1.

Note that the starting point is assumed to be valid, so it might not be included in the bank.

Example 1:

Input:
startGene = "AACCGGTT",
endGene = "AACCGGTA",
bank = ["AACCGGTA"]

Output: 1

Understanding the Problem

If there was no valid gene constraint:
We'd need one mutation for each different character
(We just need to calculate how many distance between start & end

$$min\_mutations = |\{i \mid start\_i \neq end\_i\}|$$

startGene = "AACCGGTT"
endGene = "AAACGGTA"

Min_mutations: 2

bank = ["AACCGGTA","AACCGCTA","AAACGGTA"]

Need to solve: *What is the "shortest path" between start and end?*

BFS vs. DFS: Choose BFS for optimal solutions in unweighted graphs.

➢ **BFS** ensures the **shortest path** to a solution by its level-by-level exploration. Why?

Reason 1: Nature of Leaf Nodes

- **The solution is located at one of the leaf nodes:**
  Leaf nodes don't have any children, so a leaf node either represents a node that isn't a solution but doesn't have any further paths of exploration (a dead end), or is in fact a solution **(the BFS returns when it hits a solution).**

Reason 2: Level-by-Level Traversal

- **A BFS traverses level-by-level, from top-down, which means that it always encounters the leaf nodes that are closest to the root first.**
  Thus, when a BFS encounters a leaf node that is a solution, the path from the root to that solution will be the shortest.

input:

start = "AACCGGTT", end = "AAACCGTA",
bank = ["AAACGGTT", "ACCCGGTT", "AACCGGGT", "AAACGGTA",
"AAACGGTC", "ACCCGGGT", "CAACGGTC", "GAACGGTC", "AAACCGTA"]

"AACCGGTT"

Select the ones in the bank
& not visited

mutations:

"AACCGGTT"	"AACCAGTT"
"CACCGGTT"	"AACCCGTT"
"GACCGGTT"	"AACCGGTT"
"TACCGGTT"	"AACCTGTT"
"AACCGGTT"	"AACCGATT"
"ACCCGGTT"	"AACCGCTT"
"AGCCGGTT"	"AACCGGTT"
"ATCCGGTT"	"AACCGTTT"
"AAACGGTT"	"AACCGGAT"
"AACCGGTT"	"AACCGGCT"
"AAGCGGTT"	**"AACCGGGT"**
"AATCGGTT"	"AACCGGTT"
"AACAGGTT"	"AACCGGTA"
"AACCGGTT"	"AACCGGTC"
"AACGGGTT"	"AACCGGTG"

queue:

"AACCGGTT"

input:
start = "AACCGGTT", end = "AAACCGTA",
bank = ["AAACGGTT", "ACCCGGTT", "AACCGGGT", "AAACGGTA",
"AAACGGTC", "ACCCGGGT", "CAACGGTC", "GAACGGTC", "AAACCGTA"]

gene:

"AACCGGTT"

queue:

"ACCCGGTT"

"AAACGGTT"

"AACCGGGT"

input:

start = "AACCGGTT", end = "AAACCGTA",
bank = ["AAACGGTT", "ACCCGGTT", "AACCGGGT", "AAACGGTA",
"AAACGGTC", "ACCCGGGT", "CAACGGTC", "GAACGGTC", "AAACCGTA"]

gene:

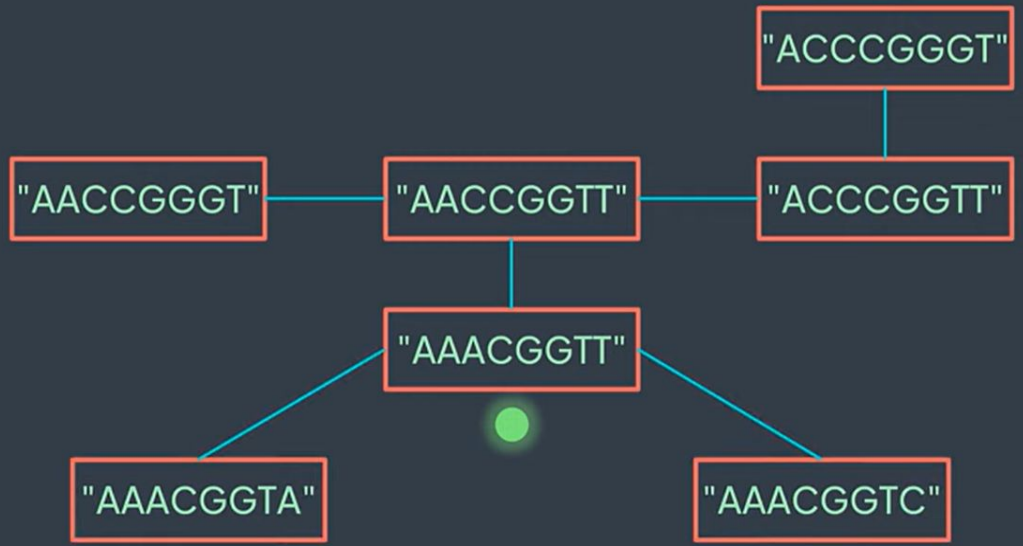
"AAACGGTT"

queue:

"AACCGGGT"

"ACCCGGGT"

input:
start = "AACCGGTT", end = "AAACCGTA",
bank = ["AAACGGTT", "ACCCGGTT", "AACCGGGT", "AAACGGTA",
"AAACGGTC", "ACCCGGGT", "CAACGGTC", "GAACGGTC", "AAACCGTA"]



gene:

"AAACGGTT"

queue:

"AACCGGGT"

"ACCCGGGT"

"AAACGGTA"

"AAACGGTC"

input:
start = "AACCGGTT", end = **"AAACCGTA"**,
bank = ["AAACGGTT", "ACCCGGTT", "AACCGGGT", "AAACGGTA",
"AAACGGTC", "ACCCGGGT", "CAACGGTC", "GAACGGTC", "AAACCGTA"]

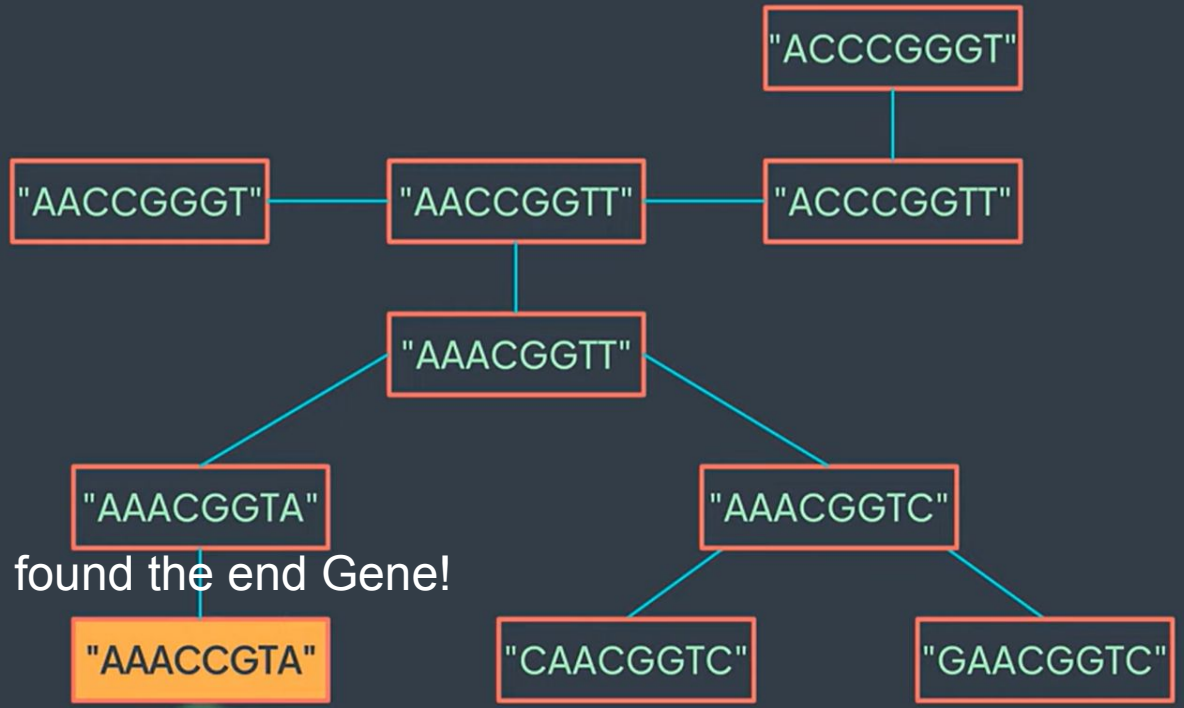
gene:

"AAACCGTA"

queue:

"CAACGGTC"

"GAACGGTC"



Code Solution

```
def minMutation_bfs(self, startGene, endGene, bank):
    queue = deque([(startGene, 0)]) # (node, count)
    seen = {startGene}

    while queue:
        node, steps = queue.popleft()
        if node == endGene:
            return steps

        # Try all possible mutations for the current gene
        for c in "ACGT":
            for i in range(len(node)): # iterate through all positions in the gene
                neighbor = node[:i] + c + node[i + 1:] # Create new mutation (replace the character)
                if neighbor not in seen and neighbor in bank:
                    queue.append((neighbor, steps + 1))
                    seen.add(neighbor)
    return -1
```

감사합니다!

THANK YOU