BFS & DFS

by

Brent Orlina

Computer Programming III

Nicholas Seward

January 30, 2024

Shortest Transformation Sequence

- Find the all of the shortest transformation sequences from one word to another
- Transformations are defined as removing, adding, or modifying a letter to the word.
- The transformation done to the word must result as another valid word in the dictionary.
- If there are multiple sequences, return all of the sequences.

Transformation Sequences for

TRAIN -> GAINS

TRAIN -> RAIN -> RAINS -> GAINS

TRAIN -> RAIN -> GAIN -> GAINS

TRAIN -> TRAINS -> GRAINS -> GAINS

TRAIN -> TRAINS -> RAINS -> GAINS

TRAIN -> GRAIN -> GRAINS -> GAINS

TRAIN -> GRAIN -> GAIN -> GAINS

Implementation

Breadth First Search was used to search through the sequences of transformations.

The starting word is `root_word`. The function `get_next_words()` gets all valid transformed words.

If the the next word was already found in some shorter sequence, it does not get appended onto the queue.

A solution sequence is if the end of the sequence is the `root_word`. However, there are multiple solution sequences with different lengths. To only keep the shortest solution sequences, the length of the first solution sequence is stored into `max_depth`. Any sequence that is longer than `max_depth` is then not considered. Since the algorithm uses BFS, solution sequence should always be the shortest.

```
def bfs(root_word: str, goal_word: str, dictionary: Set):
    Q = deque([tuple([root])])
    visited = {root: 0}
    solutions = set()

    max_depth = None
    while Q:
        sequence = Q.popleft()
        depth, last_word = len(sequence)-1, sequence[-1]

        for next_word in get_next_words(last_word, dictionary):
            if max_depth and depth+1 > max_depth:
                continue

            if next_word in visited and visited[next_word] < depth+1:
                continue

            new_sequence = (*sequence, next_word)

            visited[next_word] = depth+1

            if last_word == goal_word:
                solutions.add(sequence)
                if not max_depth:
                    max_depth = depth
                continue

            Q.append(new_sequence)

    return solutions, max_depth
```

Challenges

I had particular trouble with finding all of the valid sequences that led to the goal word. I would only find one sequence even if there were multiple solution sequences.

I was having this issue because I was keeping track of if words were already visited. This means that if word `A` were transformed to some word `C`, then word `B`, could not be transformed to word `C`, since it was already visited during word `A`.

I fixed this by keeping track of at what depth a word is first visited. If it encounters that word in any depth, it is invalid since it could have been visited at an earlier depth.

```
-- visited = set([root_word])
++ # {word: depth word was first found}
++ visited = {root_word: 0}

...

-- if next_word in visited:
++ if (next_word in visited and
++     visited[next_word] < depth+1):
```

Results

Shortest Transformation Sequences for

FREAK -> CAKE

FREAK -> FREAD -> READ -> RAD -> RAE -> RAKE -> CAKE

FREAK -> FREAD -> FRED -> FRE -> CRE -> CARE -> CAKE

FREAK -> FREAD -> FRED -> FRE -> FARE -> CARE -> CAKE

FREAK -> FREAD -> FRED -> FRE -> ARE -> CARE -> CAKE

FREAK -> FREAD -> FRED -> FRE -> FARE -> FAKE -> CAKE

Average Time Taken: 1.9s

Number of Pop Operations: 46145

Boggle

- Find all of the words starting from some letter on the board.
- All letters in the word must be connected from one letter to the next.
- A letter on the board can only be used once for a particular word.

Implementation

Depth First Search was used to search from a starting letter in the board and finding all of the words starting with that particular letter.

All the possible coordinates on the board are used as the roots in finding the words starting from the root.

Then the surrounding coordinates within the board of that coordinate is used to create a longer sequence.

If that coordinate is already the sequence, that new sequence is thrown away. If that new sequence is not a prefix of any of the words in the dictionary, it is thrown away. If that sequence is in the dictionary, then the word represented by the sequence is added to the set of found words.

```
def dfs(board: List, dictionary: Set):
    prefixes = set(w[:i+1] for w in dictionary for i in range(len(w)))

    Q = deque([tuple([(r,c)]) for r in range(len(board)) for c in range(len(board[r]))

    found_words = {}

    while Q:
        sequence = Q.pop()

        last_coord = sequence[-1]
        word = ''.join(board[coord[0]][coord[1]] for coord in sequence)

        if word in dictionary:
            found_words[word] = sequence

        for edge in get_edges(last_coord):
            if edge in sequence:
                continue

            r, c = edge
            if not (0 <= r <= len(board)-1 and 0 <= c <= len(board[r])-1):
                continue

            new_sequence = (*sequence, edge)
            new_word = ''.join(board[coord[0]][coord[1]] for coord in new_sequence)

            if new_word not in prefixes:
                continue

            Q.append(new_sequence)

    return found_words
```

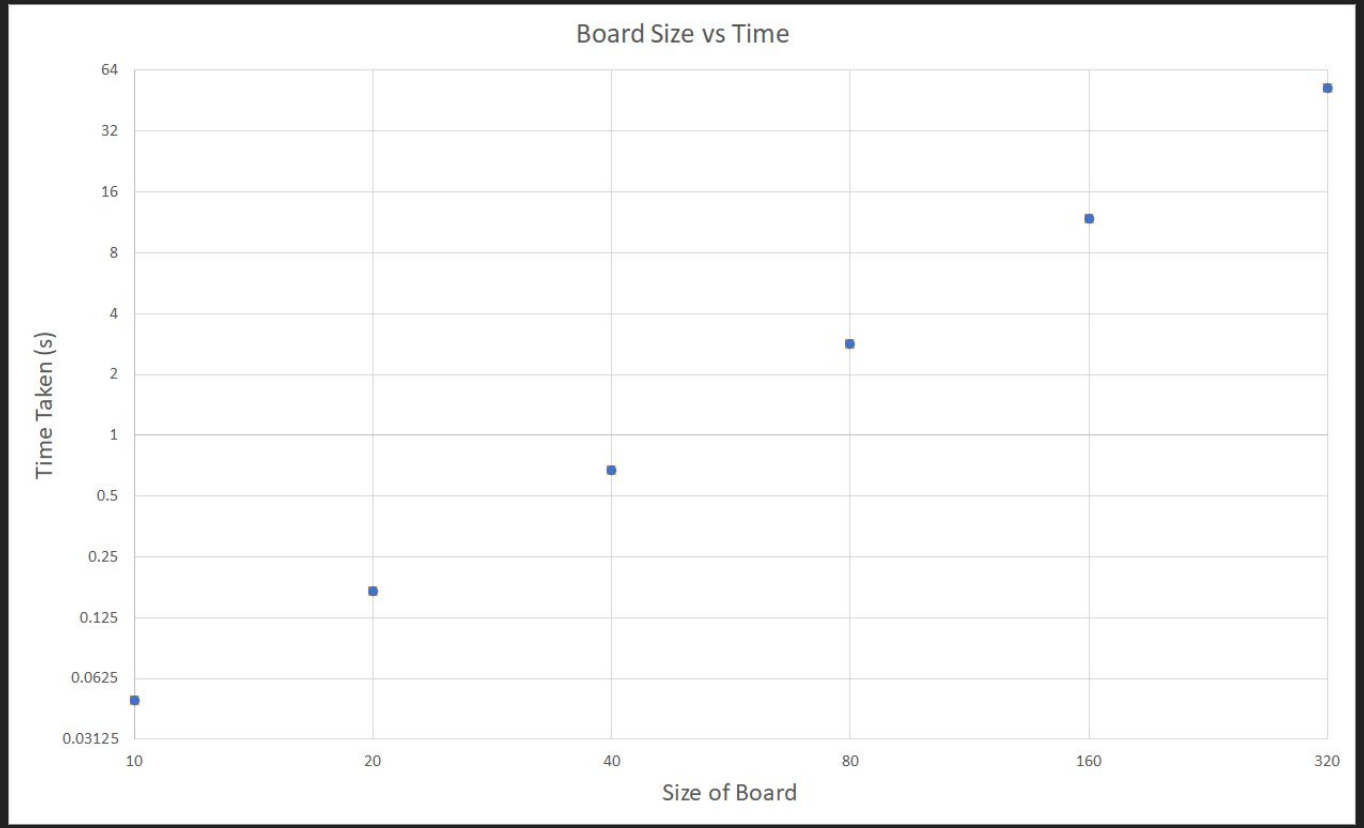
Results

Seed Used: 649634

Word Count: 1223

Longest Word: LISTENERS

Results: Board Size vs Time



Results: Board Size vs Deque Size