BFSDFS

Jayden Patel
CP3
1/30/24

Part 1 - Breadth First Search

Part 1 tasked us to write a program that takes two words as input and finds the shortest chains of transformations that get you from the start word to the target word. A transformation is defined as either deleting one character, adding one character, or modifying one character. Additionally, the program should also report how many pop operations were performed on the queue to find all the shortest chains.

Getting The Next Word

A generator function was created that returns every new possible transformation from an input word and a set of letters to pick from.

```
def get_transformations(word, haystack):
    # Remove letters
    for i in range(len(word)):
        yield word[:i] + word[i+1:]
    # Change letters
    for i in range(len(word)):
        for new_letter in haystack:
            if new_letter != word[i]:
                yield word[:i] + new_letter +
word[i+1:]
    # Add letters
    for i in range(len(word) + 1):
        for letter in haystack:
            yield word[:i] + letter + word[i:]
```

OX	UOX	FRX	FON	KFOX	FGOX	FOCX	FOYX	FOXU
FX	VOX	FSX	FOO	LFOX	FHOX	FODX	FOZX	FOXV
FO	WOX	FTX	FOP	MFOX	FIOX	FOEX	FOXA	FOXW
AOX	XOX	FUX	FOQ	NFOX	FJOX	FOFX	FOXB	FOXX
BOX	YOX	FVX	FOR	OFOX	FKOX	FOGX	FOXC	FOXY
COX	ZOX	FWX	FOS	PFOX	FLOX	FOHX	FOXD	FOXZ
DOX	FAX	FXX	FOT	QFOX	FMOX	FOIX	FOXE	
EOX	FBX	FYX	FOU	RFOX	FNOX	FOJX	FOXF	
GOX	FCX	FZX	FOV	SFOX	FOOX	FOKX	FOXG	
HOX	FDX	FOA	FOW	TFOX	FPOX	FOLX	FOXH	
IOX	FEX	FOB	FOY	UFOX	FQOX	FOMX	FOXI	
JOX	FFX	FOC	FOZ	VFOX	FROX	FONX	FOXJ	
KOX	FGX	FOD	AFOX	WFOX	FSOX	FOOX	FOXK	
LOX	FHX	FOE	BFOX	XFOX	FTOX	FOPX	FOXL	
MOX	FIX	FOF	CFOX	YFOX	FUOX	FOQX	FOXM	
NOX	FJX	FOG	DFOX	ZFOX	FVOX	FORX	FOXN	
OOX	FKX	FOH	EFOX	FAOX	FWOX	FOSX	FOXO	
POX	FLX	FOI	FFOX	FBOX	FXOX	FOTX	FOXP	
QOX	FMX	FOJ	GFOX	FCOX	FYOX	FOUX	FOXQ	
ROX	FNX	FOK	HFOX	FDOX	FZOX	FOVX	FOXR	
SOX	FPX	FOL	IFOX	FEOX	FOAX	FOWX	FOXS	
TOX	FQX	FOM	JFOX	FFOX	FOBX	FOXX	FOXT	

Output of get_transformations(“fox”)

Searching

The search loop takes the first chain in the queue for each run. Since the first chains that are created are pulled out first, it is a BFS algorithm. Each transformation from the previous function is checked against the 20k most common words list. If it is a real word and the word has not been found previously in a shorter chain, the depth is saved and the new chain is added to the queue. When a chain is at the end word, they are saved in a separate list. There is a hard chain length limit of six words; any chains longer than six words are skipped.

```
while queue:
    path = queue.pop(0)
    pops += 1
    if len(path) >= 6:
        continue
    for transformation in get_transformations(path[-1], letters):
        if transformation in words:
            if transformation in found_words:
                if len(path) > found_words[transformation]:
                    continue
            else:
                found_words[transformation] = len(path)

            new_path = path.copy()
            new_path.append(transformation)
            if transformation == end:
                paths.append(new_path)
                continue
            queue.append(new_path)
```

Results

The following paths were found in 218.35 seconds with 70,789 pop operations (loop iterations):

- ['FOX', 'BOX', 'BOD', 'BOND', 'BOUND', 'HOUND']
- ['FOX', 'BOX', 'BON', 'BOND', 'BOUND', 'HOUND']
- ['FOX', 'FOR', 'FORD', 'FOND', 'FOUND', 'HOUND']
- ['FOX', 'FOO', 'FOOD', 'FOND', 'FOUND', 'HOUND']

A major bug that arose after running the program was that the fourth path was not being found. This was because the get_transformations function used str.replace, which replaced all instances of a letter with a new letter when returning the changed-letter transformations. For example, when the function replaced the first O in food with another letter, it would replace both, producing transformations like “faad”, “fbbd”, “fccd”, etc. After switching to the current approach, it strictly replaced the letter in each index independently, which allowed “fond” to be created from “food” and the last path to be found.

Part 2 - Depth First Search

Part 2 tasked us to find every possible word from a randomly generated 10x10 letter board. The program would have to create paths starting from every letter and continuingly branch to neighboring letters to see if a real word is forming. The program should track how many words are found and what the longest word is.

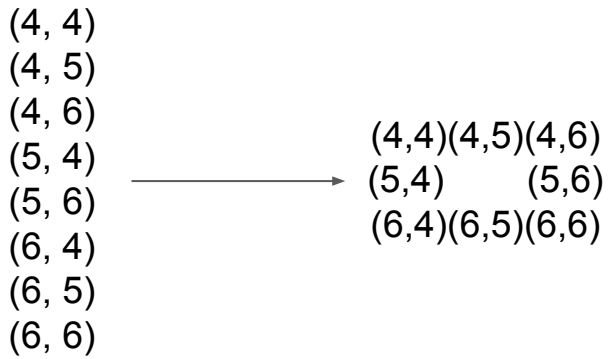
```
board = [['I', 'C', 'A', 'N', 'T', 'P', 'O', 'A', 'L', 'E'],
         ['N', 'W', 'Y', 'T', 'M', 'L', 'I', 'I', 'S', 'I'],
         ['H', 'R', 'D', 'I', 'I', 'S', 'I', 'T', 'S', 'U'],
         ['R', 'T', 'G', 'T', 'O', 'A', 'A', 'E', 'G', 'I'],
         ['S', 'R', 'G', 'I', 'I', 'E', 'I', 'R', 'A', 'C'],
         ['E', 'N', 'Y', 'S', 'E', 'S', 'P', 'E', 'E', 'X'],
         ['N', 'I', 'T', 'Y', 'N', 'S', 'E', 'N', 'R', 'R'],
         ['B', 'E', 'T', 'G', 'T', 'B', 'R', 'U', 'O', 'E'],
         ['N', 'E', 'S', 'N', 'U', 'I', 'N', 'C', 'E', 'U'],
         ['O', 'D', 'S', 'A', 'O', 'L', 'U', 'O', 'I', 'S']]
```

Seed: 6923313

Getting The Next Letter

A generator function was created that returns every possible neighbor from an input position, making sure it does not return positions off the board. The search loop finds the letter in each of the output positions from get_neighbors.

Output of get_neighbors(5,5):




```
def get_neighbors(x,y):
    for i in range(-1,2):
        for j in range(-1,2):
            if (i,j) == (0,0):
                continue
            if 0 <= x+i < 10 and 0 <= y+j < 10:
                yield (x+i,y+j)
```

Setting up the Partials

A comprehensive list of every slice of every length of word in the 20k most common words list. For example, with the word “orange”, every partial will be:

- o
- or
- ora
- oran
- orang

This was repeated for every word in the word list. As paths are created later, they can be checked against this list to see if a real word is forming.

Searching

The search loop takes the last path in the stack for each run. Since the last paths that are created are pulled out first, it is a DFS algorithm. Positions from the previous function were skipped if they already were in the path. The letter in each new position was added to the path, and the path is checked to see if it was a valid partial of a real word. Valid paths are added to the stack, and paths with full words are saved.

```
while stack:
    path = stack.pop()
    word = ""
    for x,y in path:
        word += board[y][x]

    if word in words:
        found_words.add(word)

    end = path[-1]
    for neighbor in get_neighbors(*end):
        if neighbor in path:
            continue
        new_path = path.copy()
        new_path.append(neighbor)
        new_word = ""
        for x,y in path:
            new_word += board[y][x]
        if new_word in partials:
            stack.append(new_path)
```

Results

1,014 words were found in 1.43 seconds, with the longest length being 8 letters (the random board seed was 6923313). The words found of length 8 are:

- SECURITY
- SETTINGS
- TUNGSTEN
- ENERGIES
- SEPERATE (a misspelled word)
- NINTENDO

BFSDFS

Jayden Patel
CP3
1/30/24