AI blinds search with Wordle

ARTIFICIAL INTELLIGENCE

CE KMITL 2022

TEAM เอ๋ไอ๋

สุธี สาระพันธ์
63015190
- Idea & Wordle Game

ภูษิต เสือโคร่ง
63015137
-Blinds

ธนวัฒน์ สุขแก้ว
63015069
-Statistics
-Tester

วายุ แสงพิทักษ์
63015161
-Open Word
-Filter Word

ศรายุทธ พ่อค้า
63015165
-BST
-Statistics

What is Wordle

Wordle พัฒนาโดย Josh Wardle วิศวกรซอฟต์แวร์ที่ทำงานในบรูคลิน สร้างเพื่อเป็นของขวัญวันเกิดให้เพื่อน ไม่ได้ตั้งใจจะให้เกมนั้นเป็นที่รู้จักในวงกว้างแต่อย่างใด จากการบอกต่อและเล่นต่อๆ กัน ทำให้ความนิยมก็แพร่กระจายไปทั่วอินเทอร์เน็ตอย่างต่อเนื่อง เหตุผลที่สนใจในwordleแล้วอยากนำมาทำเป็นเกมก็คือ ด้วยตัวสมาชิกภายในกลุ่มทุกคนชอบเล่นเกมเป็นทุนเดิมอยู่แล้วและได้ไปเห็นไอเดียการทำเกมต่างๆในinternetจนมาเจอwordleจึงเกิดความสนใจที่จะทำเกมนี้ขึ้นมา

How to Play Wordle

วิธีการเล่น: มีตารางใส่คำศัพท์ 5 ตัวอักษร ผู้เล่นกดที่แป้นพิมพ์ตัวอักษรเพื่อทายคำ เมื่อครบ 5 ตัวอักษรแล้วกด Enter

- ตัวอักษรนั้นเป็นตัวอักษรที่ใช่และอยู่ถูกตำแหน่ง ตัวอักษรจะกลายเป็นสีเขียว S
- เป็นตัวอักษรที่ใช่ แต่ไม่ถูกตำแหน่ง ตัวอักษรจะกลายเป็นสีเหลือง L
- ถ้าเป็นตัวอักษรที่ไม่ใช่เลย จะกลายเป็นสีเทา A
- เพื่อไม่ให้เป็นการสปอยล์คนอื่น รูปแบบการแชร์ของ Wordle จะแสดงแค่แพทเทิร์นสีและเลขคะแนนเท่านั้น มองไม่เห็นคำและตัวอักษร เป็นการกระตุ้นให้คนอื่นอยากรู้อยากเห็นและเข้าไปเล่นตาม

HOW TO PLAY

Guess the **WORDLE** in 6 tries.

After each guess, the color of the tiles will change to show how close your guess was to the word.

The letter **W** is in the word and in the correct spot.

The letter **L** is in the word but in the wrong spot.

The letter **U** is not in the word in any spot.


Create Wordle

Words in Wordle

Conceptually and stylistically, the game is similar to the 1955 pen-and-paper game Jotto and to the game show franchise *Lingo*.[7][8][9][10] The gameplay is also similar to the two-player board game *Mastermind*—which had a word-guessing variant *Word Mastermind*[11]—and the game Bulls and Cows, with the exception that *Wordle* confirms the specific letters that are correct.[12][13][14] Each daily game uses a word from a randomly ordered list of 2,315 words (out of the approximate 12,000 five-letter words in the English language).[12][15][16] The smaller word list was chosen by Wardle's partner, who categorized the five-letter words into those she knew, those she did not know, and those she might have known.[17] *Wordle* uses American spelling, despite the developer being from Wales and using a UK domain name for the game; he is a long-time resident of Brooklyn, New York. Players outside the US have complained that this spelling convention gives American players an unfair advantage, for example in the case of "favor".[18][19][20][21]

ใน Wordle จะมีคำศัพท์ที่เป็นไปได้อยู่ 2315 คำศัพท์เราจึงทำการเก็บรวบรวมแล้วบันทึกไว้เพื่อใช้สำหรับสร้างตัวเกม Wordle ของเราเองในการทดลองใช้ Blinds search ในการแก้ปัญหา

1	eight	2249	harpy
2	dryly	2250	dizzy
3	belle	2251	ranch
4	album	2252	every
5	polka	2253	claim
6	train	2254	chuck
7	whiff	2255	fungi
8	anger	2256	forum
9	sword	2257	gaily
10	abled	2258	chore
11	hotly	2259	brisk
12	cutie	2260	moist
13	whirl	2261	poker
14	stone	2262	briar
15	eerie	2263	slunk
16	mucus	2264	crash
17	ruddy	2265	shrug
18	wield	2266	digit
19	showy	2267	shark
20	salvo	2268	whack
21	their	2269	zesty
22	serve	2270	noose
23	fleck	2271	marsh
24	adorn	2272	cough
25	rehab	2273	thick
26	china	2274	aside
27	bloat	2275	pithy
28	gaunt	2276	green
29	eclat	2277	shoal
30	chaff	2278	thong
31	talon	2279	queue
32	venom	2280	psalm
		2281	cobra
		2282	wrung
		2283	medic
		2284	quack
		2285	prior
		2286	stole
		2287	shirt
		2288	riser
		2289	bison
		2290	stall
		2291	dross
		2292	belch
		2293	tatty
		2294	being

Wordle Game

ในตัวเกมที่เราได้จำลองสร้างมาใช้ภาษา Python ในการเขียนการเช็คเงื่อนไข และ การทำงานต่าง ๆ ของตัวเกมให้คล้ายคลึงกับตัว Wordle จริง ๆ เพื่อใช้ในการทดลอง Blinds Search

01.

Opener Word

Find the best opener word to start the game.

02.

Filtered Word

Find a possible word from opening word result.

03.

Convert to BST

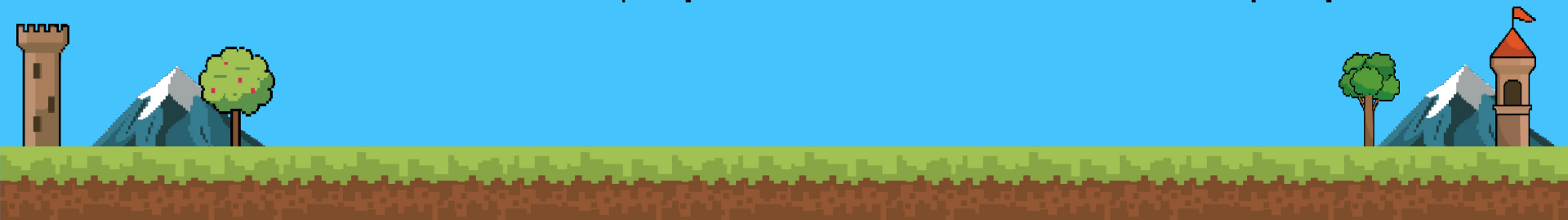
Convert possible word to Binary Search Tree(BST)

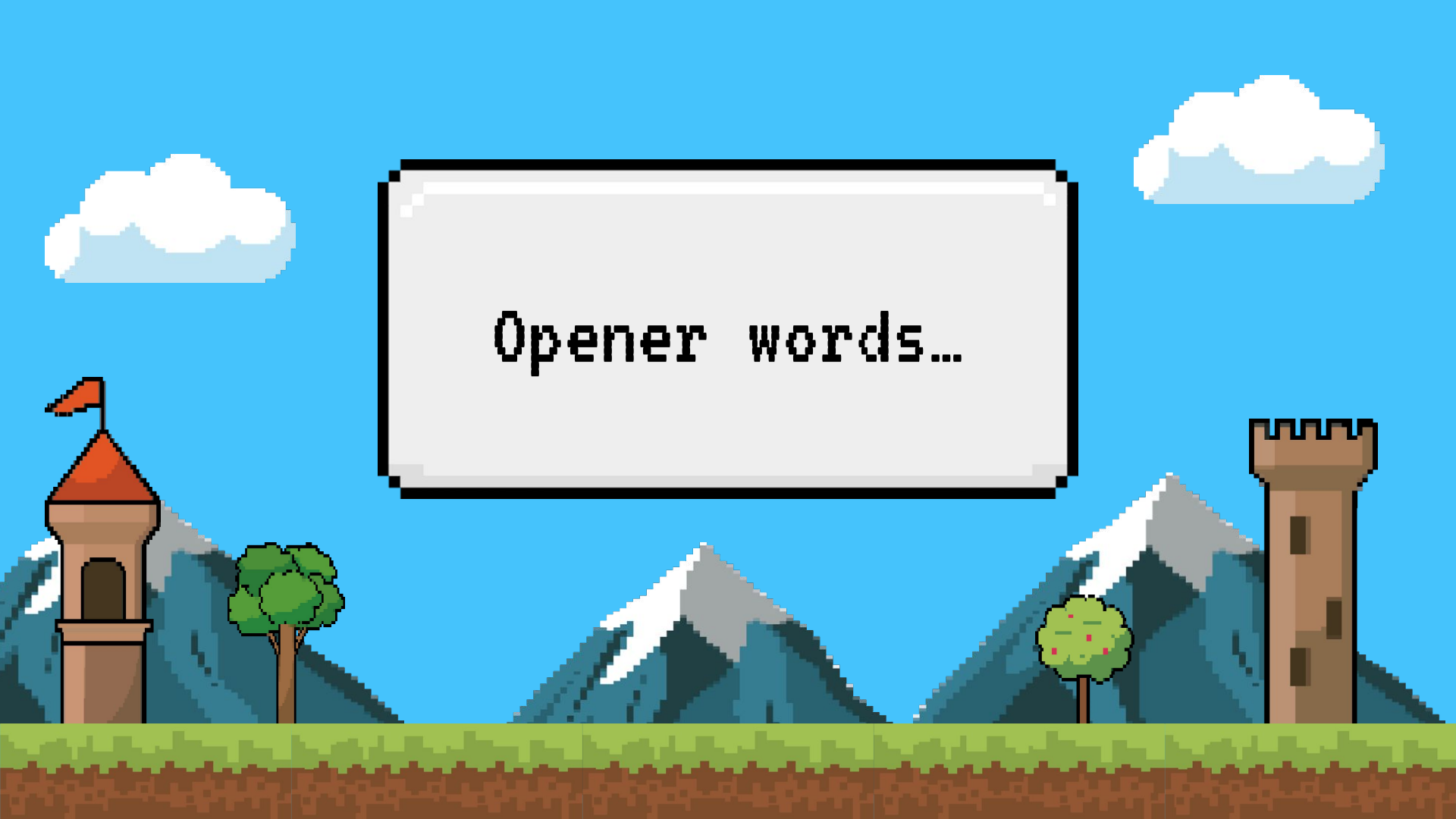
04.

Blind Search

Finding Goal from BST by using Blind Search

Opener Word → Filtered Word → Convert to BST → Blind Search




Opener words…

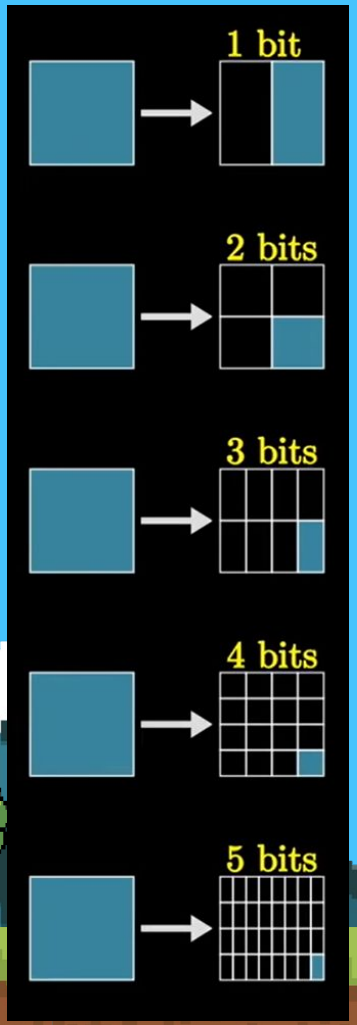
Opener words using information theory

By using this formula for every pattern of each word.

$$E[I] = \sum_x p(x) \cdot \log_2 \left((1/p(x)) \right)$$

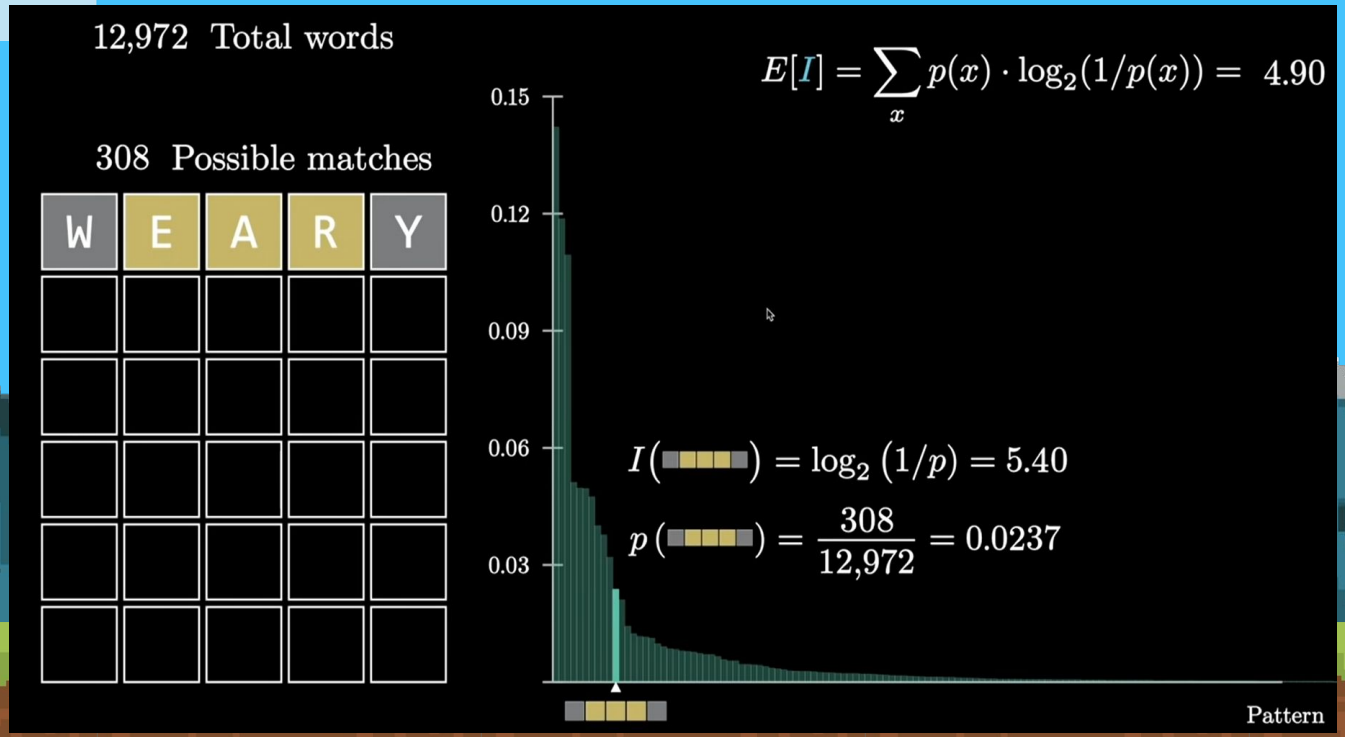
Called entropy

This mean how many times you've cut down your possibility in half.



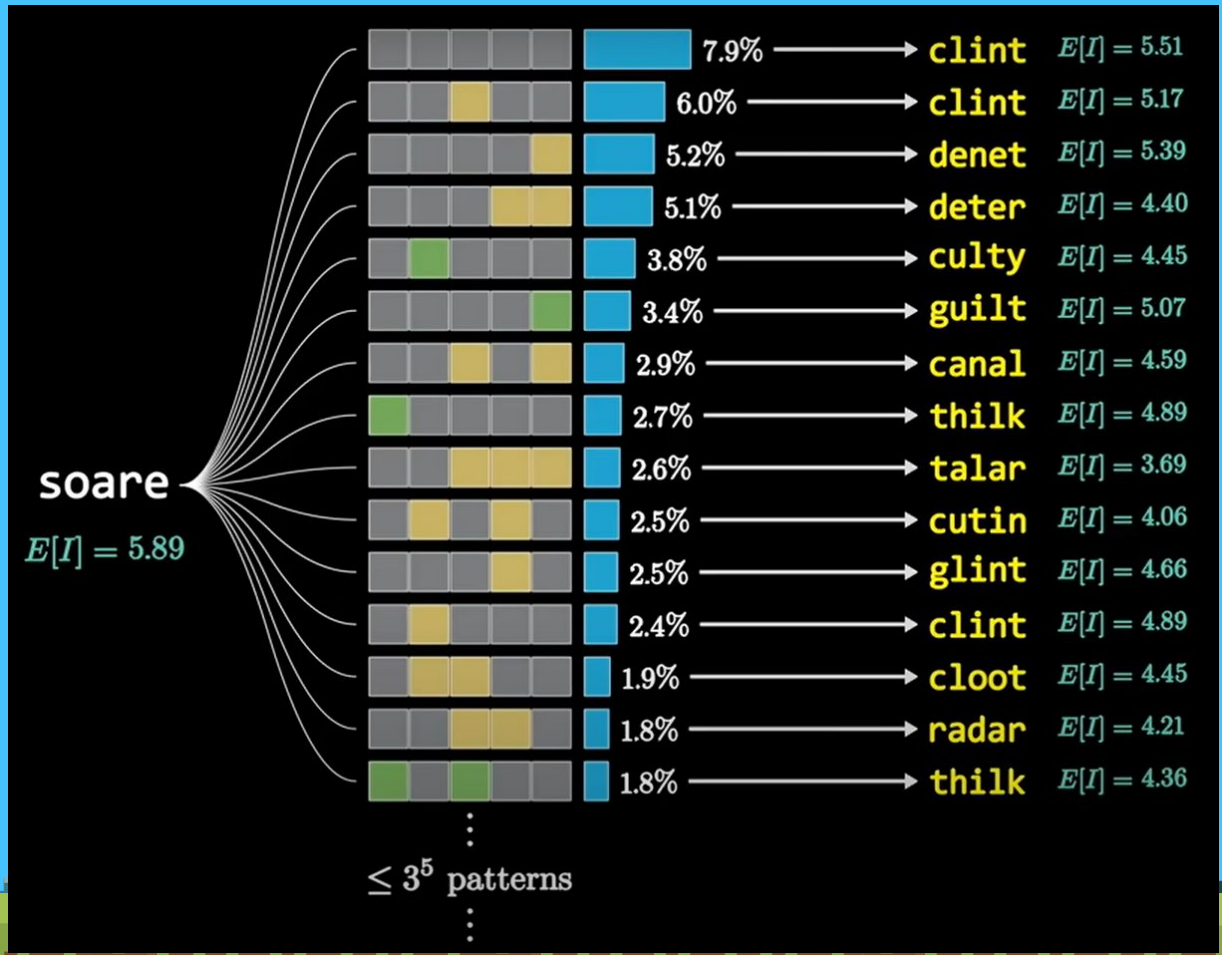
Example

Combines all patterns score of “weary” word the average is about 4.90 or cut down the possibility by about 5 times



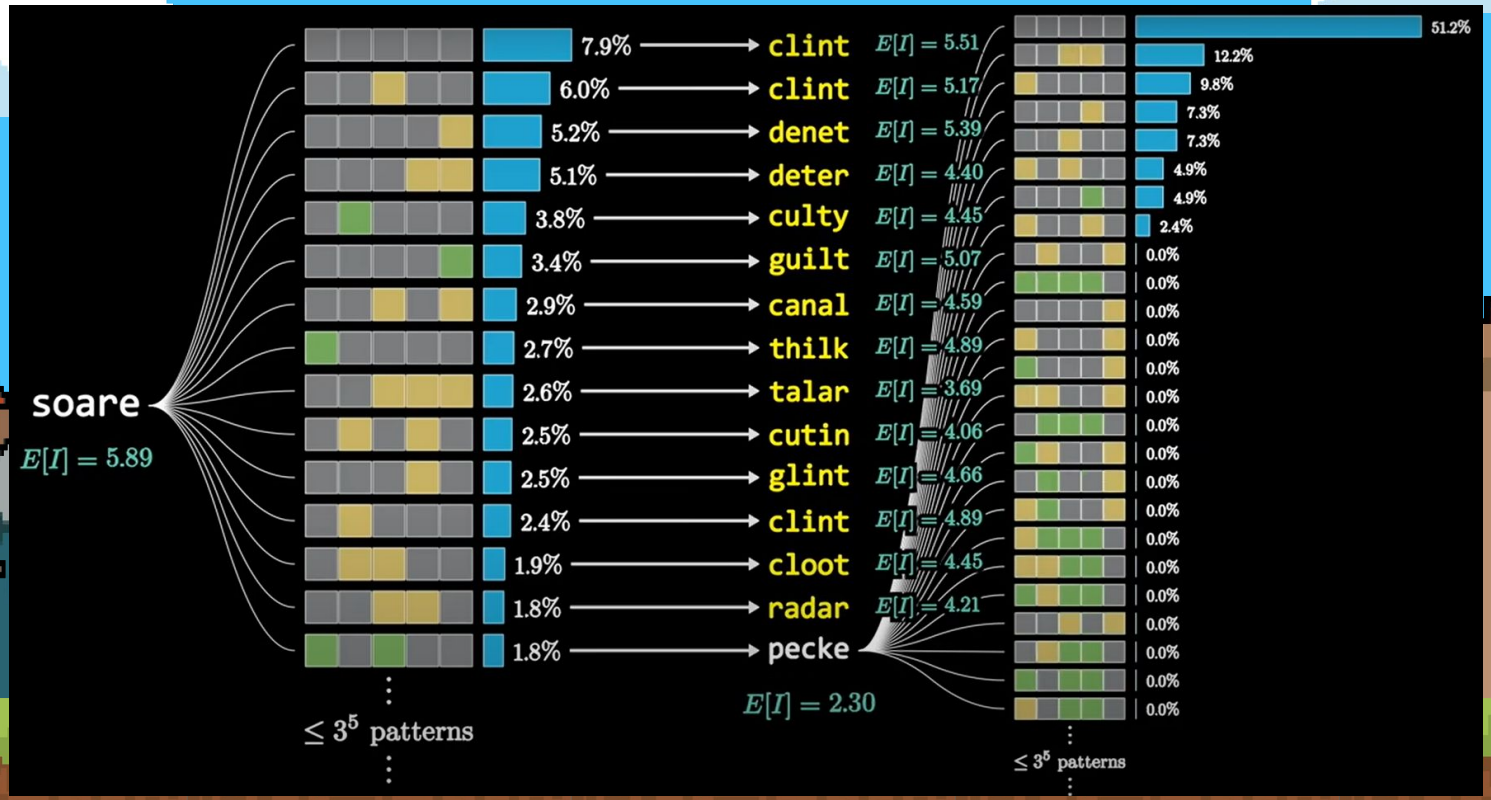
1 Step

The best 1-step is the word “soare”



Improving the answer

By repeat and do this for all of the different possible patterns



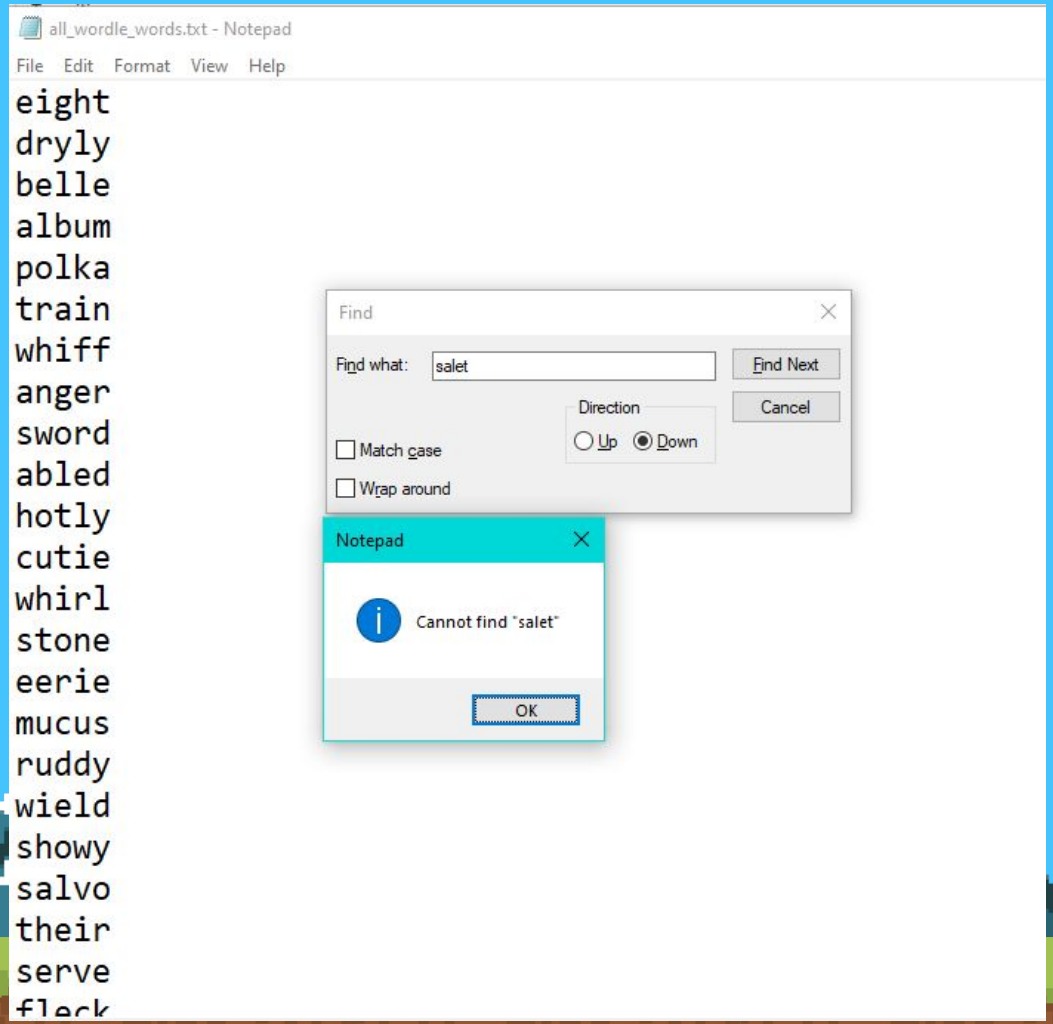
Result

So according to the result the best opener is “salet”

Highest $E[I]$ (one step)	Highest $E[I]$ (two steps)	Lowest average scores
1. soare → 5.89	1. slane → 5.77 + 4.27 = 10.04	1. salet → 3.433
2. roate → 5.88	2. slate → 5.86 + 4.18 = 10.03	2. crate → 3.434
3. raise → 5.88	3. salet → 5.83 + 4.18 = 10.02	3. trace → 3.435
4. raile → 5.87	4. trace → 5.83 + 4.18 = 10.01	4. slate → 3.436
5. reast → 5.87	5. crate → 5.83 + 4.18 = 10.01	5. reast → 3.438
6. slate → 5.86	6. reast → 5.87 + 4.14 = 10.01	6. crane → 3.438
7. crate → 5.83	7. carle → 5.77 + 4.24 = 10.01	7. slane → 3.439
8. salet → 5.83	8. roast → 5.65 + 4.35 = 10.00	8. slant → 3.440
9. irate → 5.83	9. torse → 5.72 + 4.27 = 10.00	9. carte → 3.441
10. trace → 5.83	10. carse → 5.77 + 4.23 = 10.00	10. carle → 3.443
11. arise → 5.82	11. carte → 5.79 + 4.20 = 10.00	11. trice → 3.447
12. orate → 5.82	12. toile → 5.69 + 4.31 = 10.00	12. least → 3.449
13. stare → 5.81	13. trone → 5.68 + 4.31 = 10.00	13. carse → 3.450
14. carte → 5.79	14. soare → 5.89 + 4.11 = 9.99	14. torse → 3.451
15. raine → 5.79	15. raile → 5.87 + 4.13 = 9.99	15. slart → 3.451

But in our word lists don't have "salet" so we choose the second best opener "crate" to be our opener

1. salet → 3.433
2. crate → 3.434




Filtered words

Example

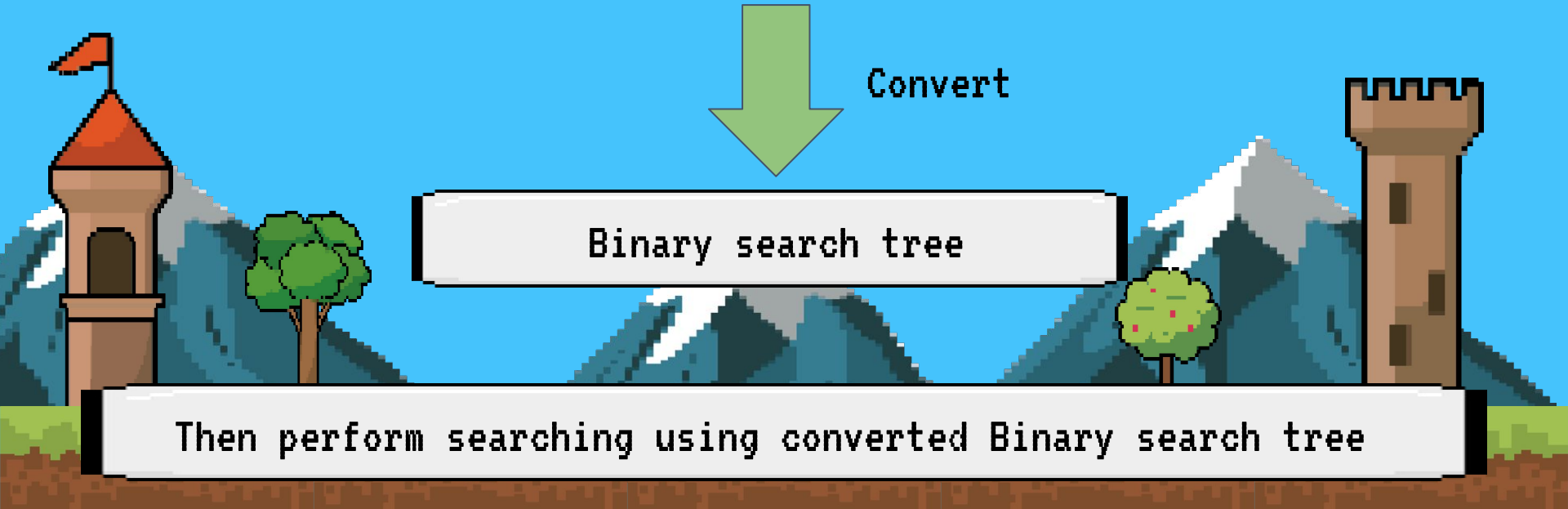
We've entered opener world "crate" and get this result

So the list of words is…

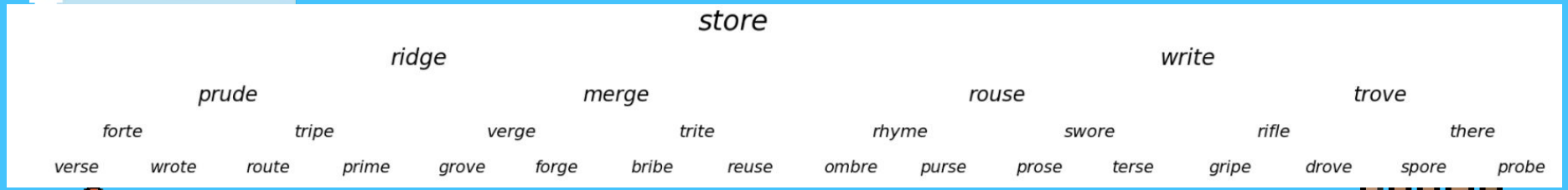
['belle', 'wield', 'venom', 'knife', 'spell', 'nosey', 'lunge', 'spend', 'bless', 'unfed', 'oxide', 'opine', 'seven', 'denim', 'kneed', 'video', 'pixie', 'beefy', 'bowel', 'seedy', 'imbue', 'pesky', 'shelf', 'pixel', 'ovine', 'welsh', 'house', 'spiel', 'index', 'ozone', 'judge', 'feign', 'lumen', 'hedge', 'wheel', 'bleed', 'noble', 'equip', 'shone', 'spoke', 'impel', 'widen', 'semen', 'devil', 'whose', 'delve', 'sense', 'wedge', 'model', 'liken', 'movie', 'enjoy', 'elude', 'poise', 'deign', 'penne', 'vixen', 'solve', 'swine', 'quell', 'slide', 'belie', 'money', 'debug', 'snide', 'jelly', 'guise', 'gooey', 'bulge', 'issue', 'phone', 'singe', 'shine', 'sleep', 'neigh', 'epoxy', 'diode', 'leggy', 'bezel', 'linen', 'goose', 'speed', 'unwed', 'geese', 'fugue', 'fudge', 'poesy', 'guile', 'glove', 'sweep', 'moose', 'louse', 'segue', 'hymen', 'swell', 'eking', 'guide', 'lemon', 'begun', 'snipe', 'biome', 'sheen', 'woven', 'eying', 'binge', 'bilge', 'newly', 'ennui', 'whine', 'edify', 'plume', 'obese', 'bevel', 'elide', 'blend', 'whelp', 'seize', 'shied', 'gouge', 'dopey', 'geeky', 'flesh', 'gnome', 'elope', 'vogue', 'booze', 'posse', 'guess', 'weigh', 'woken', 'doze

Example

['belle', 'wield', 'venom', 'knife', 'spell', 'nosey', 'lunge', 'spend', 'bless', 'unfed', 'oxide', 'opine', 'seven', 'denim', 'kneed', 'video', 'pixie', 'beefy', 'bowel', 'seedy', 'imbue', 'pesky', 'shelf', 'pixel', 'ovine', 'welsh', 'house', 'spiel', 'index', 'ozone', 'judge', 'feign', 'lumen', 'hedge', 'wheel', 'bleed', 'noble', 'equip', 'shone', 'spoke', 'impel', 'widen', 'semen', 'devil', 'whose', 'delve', 'sense', 'wedge', 'model', 'liken', 'movie', 'enjoy', 'elude', 'poise', 'deign', 'penne', 'vixen', 'solve', 'swine', 'quell', 'slide', 'belie', 'money', 'debug', 'snide', 'jelly', 'guise', 'gooey', 'bulge', 'issue', 'phone', 'singe', 'shine', 'sleep', 'neigh', 'epoxy', 'diode', 'leggy', 'bezel', 'linen', 'goose', 'speed', 'unwed', 'geese', 'fugue', 'fudge', 'poesy', 'guile', 'glove', 'sweep', 'moose', 'louse', 'segue', 'hymen', 'swell', 'eking', 'guide', 'lemon', 'begun', 'snipe', 'biome', 'sheen', 'woven', 'eying', 'binge', 'bilge', 'newly', 'ennui', 'whine', 'edify', 'plume', 'obese', 'bevel', 'elide', 'blend', 'whelp', 'seize', 'shied', 'gouge', 'dopey', 'geeky', 'flesh', 'gnome', 'elope', 'vogue', 'booze', 'posse', 'guess', 'weigh', 'woken', 'doze



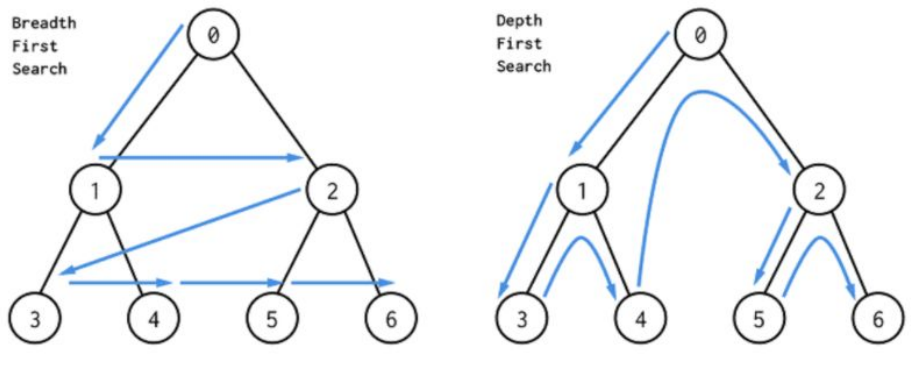
Example



Then perform searching using converted Binary search tree

Blind Search

Blind Search



Breadth First Search

Breadth First Search (BFS) is optimal for finding the shortest distance, and is more suitable for searching vertices which are closer to the given source.

Depth First Search

Depth First Search (DFS) is more suitable when there are solutions away from source. DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.

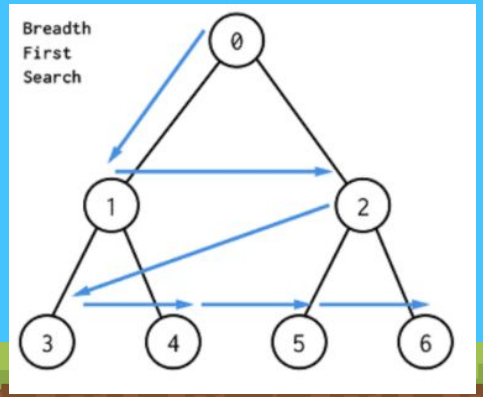
Breadth First Search

Implement BFS

```
#---------------BFS(breadth first search)-----------------
bfslist = []
def print_given_level(root, level):
    if root is None:
        return
    if level == 1:
        bfslist.append(root.data)
    elif level > 1 :
        print_given_level(root.left , level-1)
        print_given_level(root.right , level-1)
def level_order(root): ### breadth first search
    h = height(root)
    for i in range(1, h+1):
        print_given_level(root, i)
```

Breadth First Search

There are basically two functions in this method. One is to print all nodes at a given level (print_given_Level), and other is to print level order traversal of the tree (level_order). printLevelorder makes use of print_given_Level to print nodes at all levels one by one starting from the root.



Breadth First Search

Goal : blond

Step : 1

Word : think

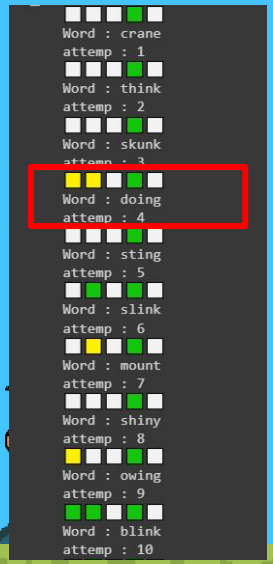
Breadth First Search

Goal : blond

Step : 2

Word : skunk

Breadth First Search

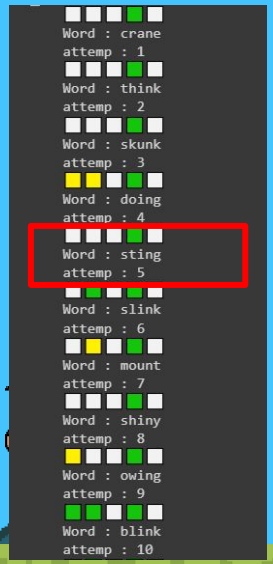


Goal : blond

Step : 3

Word : doing

Breadth First Search

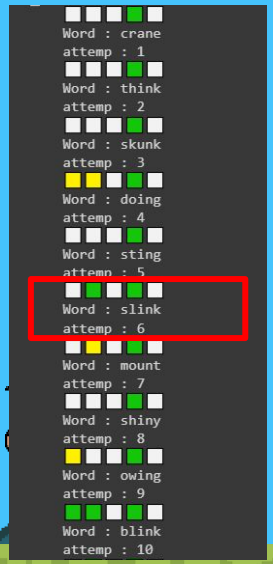


Goal : blond

Step : 4

Word : sting

Breadth First Search

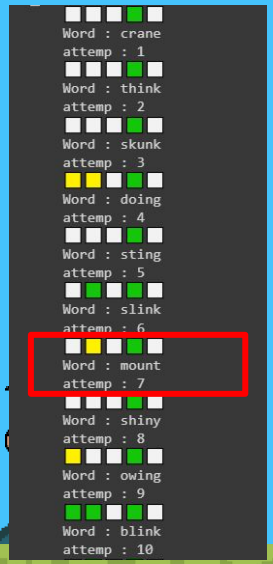


Goal : blond

Step : 5

Word : slink

Breadth First Search

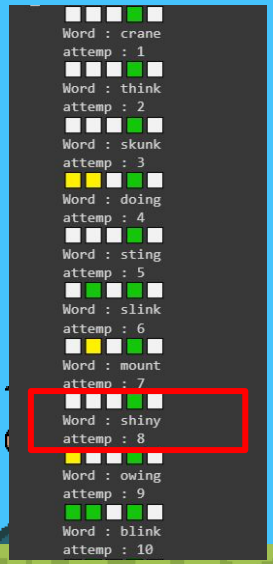


Goal : blond

Step : 6

Word : mount

Breadth First Search



Goal : blond

Step : 7

Word : shiny

Breadth First Search

Goal : blond

Step : 8

Word : owing

Breadth First Search

Goal : blond

Step : 36

Word : blond

GOAL!!

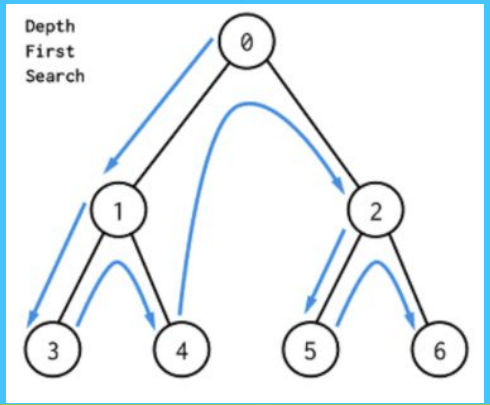
Depth First Search

Implement DFS

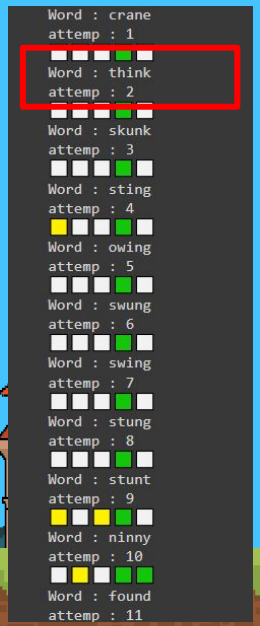
```
#--------------------DFS(Depth First Search)-------------------------
dfslist = []
def dfs(root):
    if root:
        #print(root.data),# print the data of node
        dfslist.append(root.data) # append data of node to list
        dfs(root.left)# recur on left child
        dfs(root.right)# recur on right child
```

Depth First Search

We used recursive and backtracking for traversal. In this traversal first the deepest node is visited and then backtracks to it's parent node if no sibling of that node exist.



Depth First Search



Goal : blond

Step : 1

Word : think

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 2

Word : skunk

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 3

Word : think

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 4

Word : owing

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 5

Word : swung

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 6

Word : swing

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 7

Word : stung

Depth First Search


```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 8

Word : stung

Depth First Search

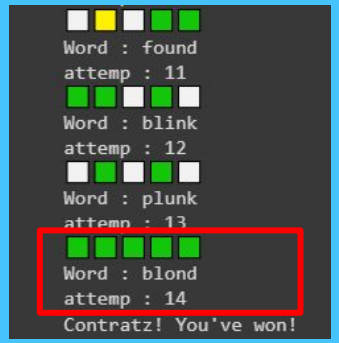

```
Word : crane
attemp : 1
Word : think
attemp : 2
Word : skunk
attemp : 3
Word : sting
attemp : 4
Word : owing
attemp : 5
Word : swung
attemp : 6
Word : swing
attemp : 7
Word : stung
attemp : 8
Word : stunt
attemp : 9
Word : ninny
attemp : 10
Word : found
attemp : 11
```

Goal : blond

Step : 9

Word : ninny

Depth First Search



Goal : blond

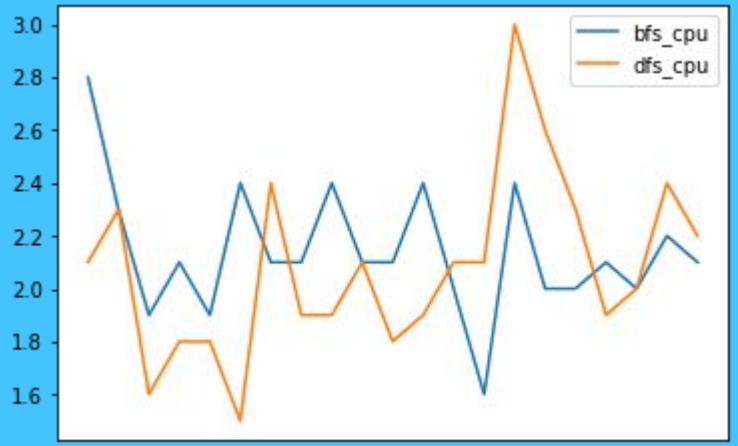
Step : 13

Word : blond

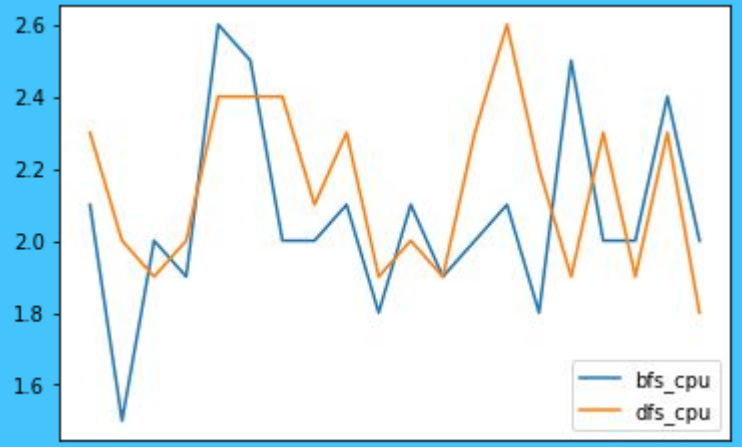
GOAL!!

Statistics

Cpu usage (N = 20 , 2 time)

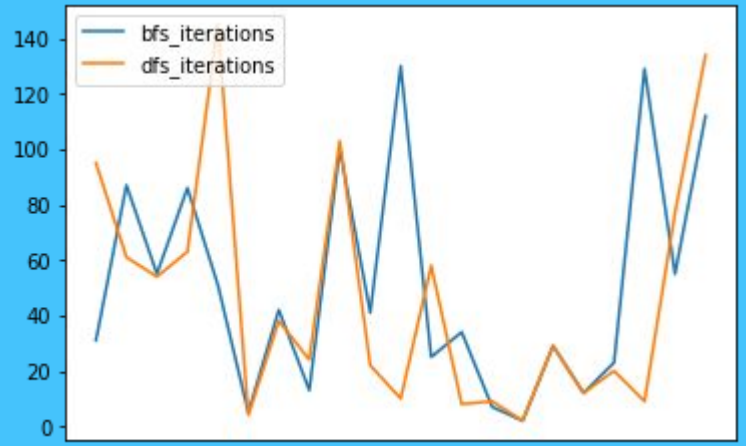


bfs cpu avr : 2.1428571428571432
dfs cpu avr : 2.080952380952381

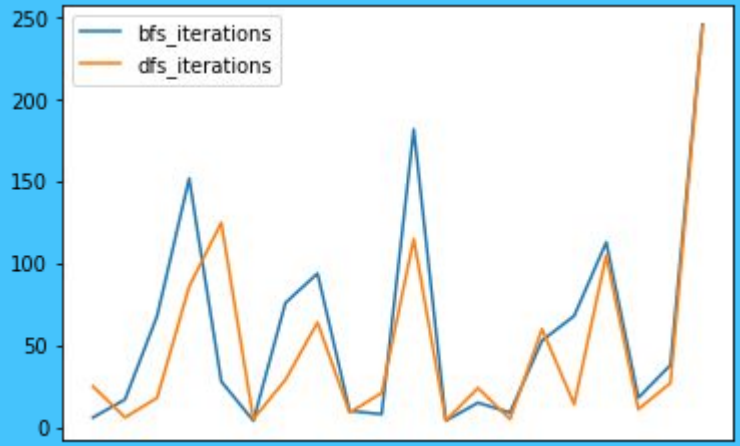


bfs cpu avr : 2.0650000000000004
dfs cpu avr : 2.1449999999999996

Iterations (N = 20 , 2 time)

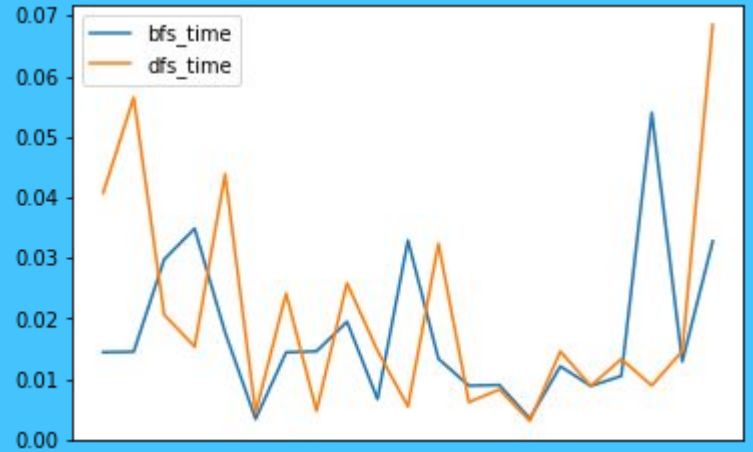


bfs iterations : 50.9
dfs iterations : 46.5

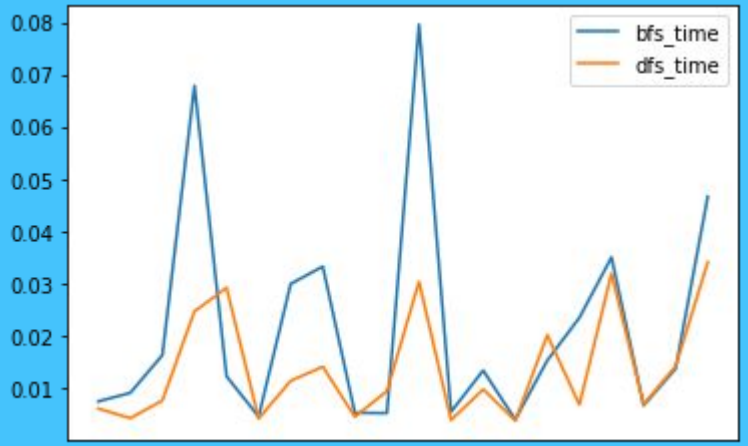


bfs iterations : 60.45
dfs iterations : 49.9

Time usage (N = 20 , 2 time)



bfs time avr : 0.017537043857146273
dfs time avr : 0.020702223380978813



bfs time avr : 0.02185764639998524
dfs time avr : 0.013992572149959415

Other rounds (N = 20)

bfs cpu avr : 2.1850000000000005
dfs cpu avr : 2.0650000000000004
bfs time avr : 0.014427315600028124
dfs time avr : 0.01144499974993778
bfs iterations : 43.7
dfs iterations : 33.65

bfs cpu avr : 2.115
dfs cpu avr : 2.0949999999999998
bfs time avr : 0.020824205950043508
dfs time avr : 0.02413814224998987
bfs iterations : 57.2
dfs iterations : 60.75

bfs cpu avr : 2.155
dfs cpu avr : 2.109999999999999
bfs time avr : 0.01957707314995787
dfs time avr : 0.018336745699980384
bfs iterations : 53.85
dfs iterations : 48.95

conclusion

- If you know a solution is not far from the root of the tree, a breadth first search (BFS) might be better.

BFS

```
Word : angel
attemp : 30
Contratz! You've won!
The CPU usage is:  5.9
RAM memory % used: 8.3
Time:  0.01551535000001536
```

DFS

```
Word : angel
attemp : 115
Contratz! You've won!
The CPU usage is:  3.8
RAM memory % used: 8.3
Time:  0.038255841999983886
```

- If you know a solution is near from left side, DFS might be better.

BFS

```
Word : pasta
attemp : 17
Contratz! You've won!
The CPU usage is:  2.9
RAM memory % used: 7.8
Time:  0.017803764000063893
```

DFS

```
Word : pasta
attemp : 6
Contratz! You've won!
The CPU usage is:  2.5
RAM memory % used: 7.8
Time:  0.011135365999962232
```


Thanks!