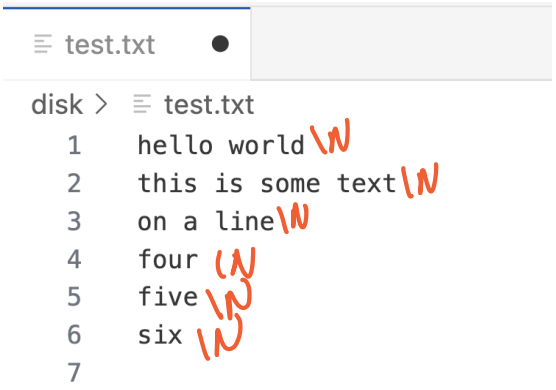
Lecture 31 – Consider the characters / Trie-ing to use data structures

Q: Can you jump to a specific line in a file? ⇒ *How do you delete a certain line in a file?*

```
# Example from HW5
DISK_PATH = pathlib.Path("disk")

def write_file():
    with open(DISK_PATH/"test.txt", "w") as f:
        f.write("hello world\n")
        f.write("this is some text\n")
        f.write("on a line\n")
        f.write("four\n")
        f.write("five\n")
        f.write("six\n")
```



A mental model for files in this class:
=> Can think about files as a list of lines, where each line is a string
=> Each line ends with a *newline character* (write as "\n"), which ends the line

This is a generally good model for working with text files, but it turns out that things are actually a bit more complicated under the hood, which complicates things like jumping to a specific line, or deleting a specific line.

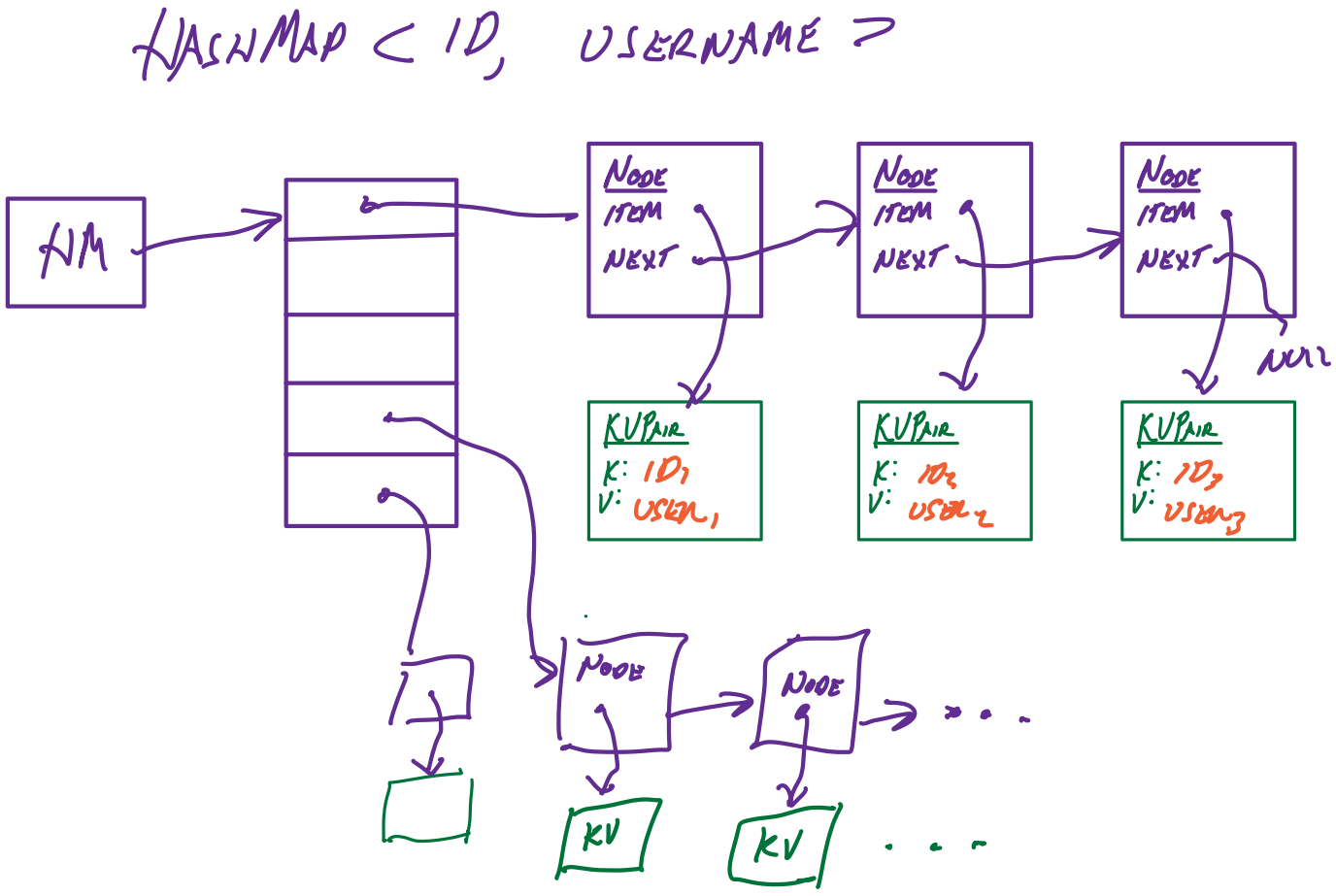
For more discussion on files (and answers to "can you jump to a specific line?" and "how to delete a line of a file?") see the end of these notes.

Design problems: Try and trie again

Design problem 1: A large-scale internet service assigns an internal ID number to each username. Here are their key criteria:

- These IDs are used for various tasks (creating URLs, storing relationships between users, etc), so lookup needs to be fast.
- The site also adds new users frequently, so assigning new IDs to new users has to be fast
- No two users can have the same ID
- Because the site has so many users, they are concerned about the space being used to store this information

What data structure do you choose? How many ~~memory addresses~~ OBJECTS would your solution require, in terms of the number U of users?



Classic hashmap structure: each array slot contains a linked list of KVPairs:

=> Number of objects: 1 HashMap object + underlying array + (1 Node + 1 KVPair)*(U users)

(Also possible to be a bit more compact and merge the Node and KVPair into one object, but structure remains the same.)

Design problem 2: Swype keyboards let people type by passing their fingers over the keys rather than hitting individual keys separately. The software for such a keyboard makes use of a dictionary of available words to help figure out which keys might be relevant, as well as which keys might come "next" based on what a user has typed so far.

https://www.flickr.com/photos/hahatango/3173033612

1. *What data structure do you use to store this dictionary?* => HashSet
2. *How will you use it to predict which keys might come next?*

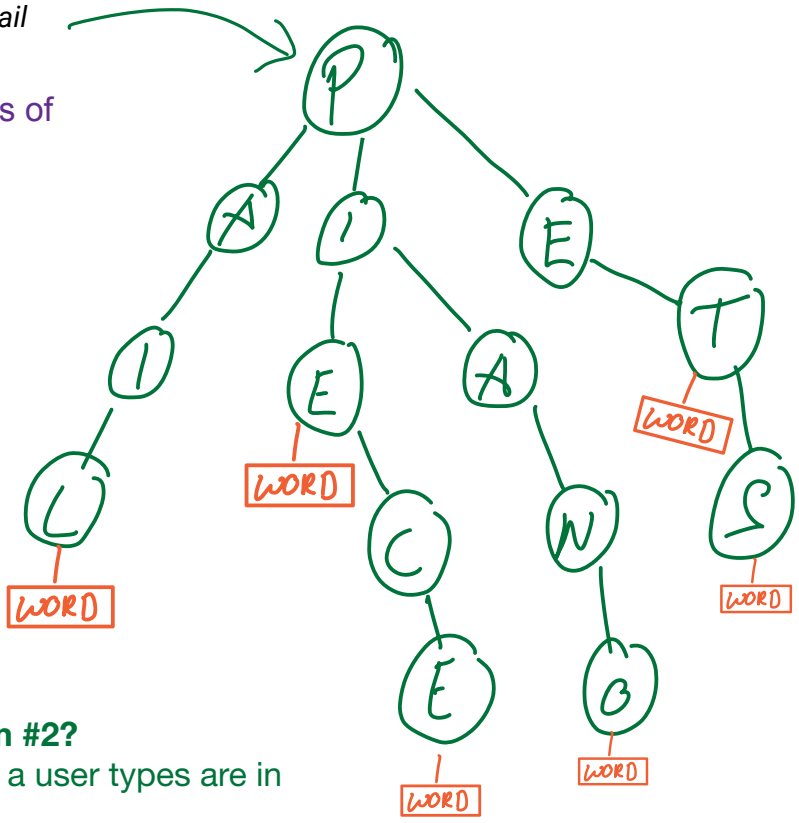
For example: How to keep all "qui" words together?
=> How to group words with a common prefix?

Could use a hashmap: prefix => (words with that prefix)
However, lots of potential prefixes:
q, qu, qui => hard to scale, because many duplicates

Can we do better???

Tries: Storing prefixes with a tree!

An example dictionary: pie, piece, piano, pet, pail



Idea: Share prefixes of words along branches of the tree
=> some nodes are "endpoints" that store a result (ie, a word in the dictionary, a decision, etc.)
=> This is called a trie (reTRIEval)

How would a trie work for design problem #2?
Can follow the trie to determine if the letters a user types are in the dictionary:
- For each letter the user enters, follow one node on the trie
- Can tell when the path ends at a valid word
- (From there, could possibly do some more computation to figure out the next "closest" word, to help with prediction)

Q: Could you embed a trie in an array?
=> Would only work if we had a defined structure (one slot for each character per branch, so that every node has the same number of child nodes)

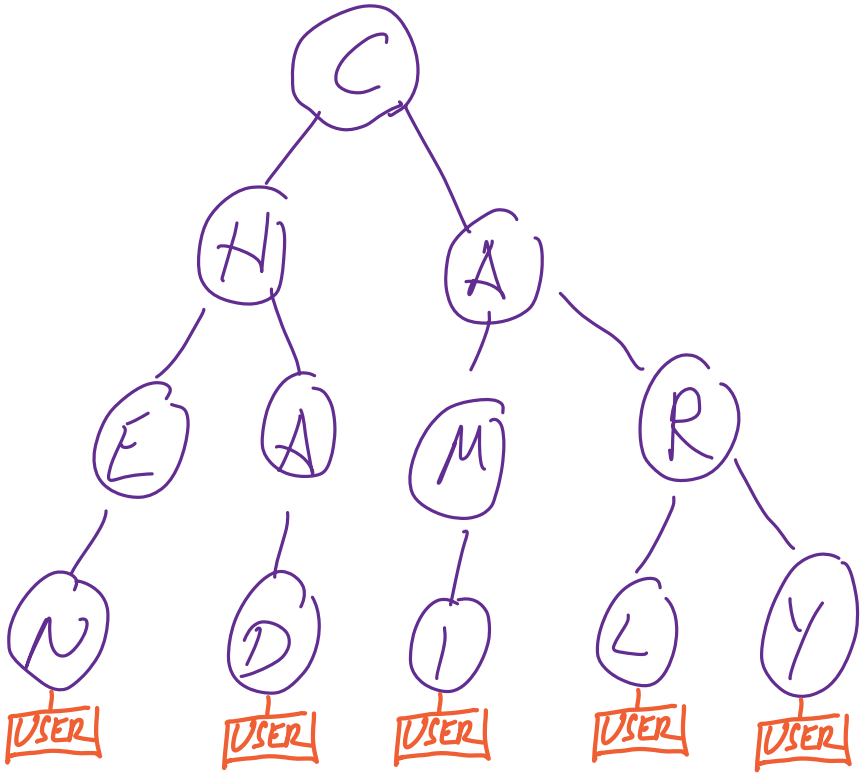
Q: Does a trie actually save space?
=> Could become very large due to the number of possible branches. However, there are some tricks to make the trie use fewer nodes (eg. reducing the long chains of nodes with no branching--not important in this course)

Back to Design Problem #1

Some usernames:

```
chen
chad
cami
carl
cary
. . .
```

What if we need to store a LOT of usernames??



Could represent usernames using a trie: time to look up a username would involve searching the tree

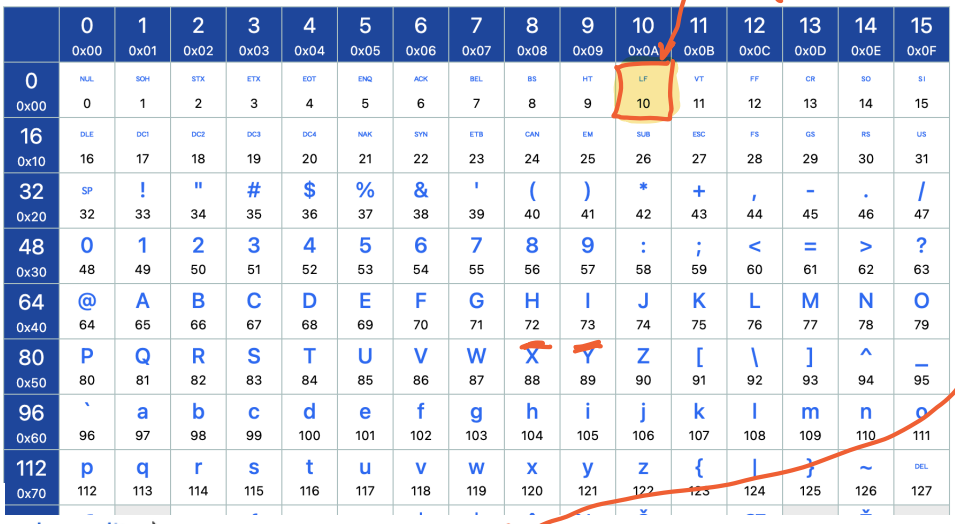
=> Even with a lot of users, the search time is based on the number of characters in the username (eg. a small, constant value, like 8 characters)

=> In some applications, this might be faster than using a hashmap, due to the time required to hash the individual characters of each username. (This tradeoff isn't something we'd ask you to make a decision about, but we want you to see the idea and understand why tries are cool.)

What's in a string? Basic version: ASCII Codes

Every character has a corresponding numeric value ("ascii code")

- "A" is 65
- "M" is 77
- "T" is 84
- "1" is 49
- "4" is 52
- "8" is 56
- …



	0 0x00	1 0x01	2 0x02	3 0x03	4 0x04	5 0x05	6 0x06	7 0x07	8 0x08	9 0x09	10 0x0A	11 0x0B	12 0x0C	13 0x0D	14 0x0E	15 0x0F
0 0x00	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
16 0x10	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
32 0x20	SP 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
48 0x30	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
64 0x40	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
80 0x50	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
96 0x60	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
112 0x70	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

NEWLINE CHARACTER! (JUST A NUMBER!)

a hex editor):

```
test.txt
disk > test.txt
1  hello world
2  this is some text
3  on a line
4  four
5  five
6  six
7
```

```
6C 6C 6F 20 77 6F 72 6C 64 0A 74 68 69 73
73 20 73 6F 6D 65 20 74 65 78 74 0A 6F 6E
20 6C 69 6E 65 0A 66 6F 75 72 0A 66 69 76
73 69 78 0A
```

```
hello world.this
is some text.on
a line.four.fiv
e.six.
```

ASCII CODES FOR THIS FILE

(You don't need to know about how these codes work, we just want you to see them.)

Every character has some numeric representation
=> A file is just a big, ordered sequence of characters

This is something you'll spend more time learning if you take a systems course (eg. CS 300 or CS 330). For CS 200, it's just helpful to know that files and strings are composed of characters, which are really just numbers.

What does this have to do with jumping to a specific line in a file?

The newline character (\n) is just another character (ie, a number). The only difference is that it has a special meaning: instead of "printing" as a letter or number, it tells the program or terminal reading the file to go to the next line.

So what does readline() do?

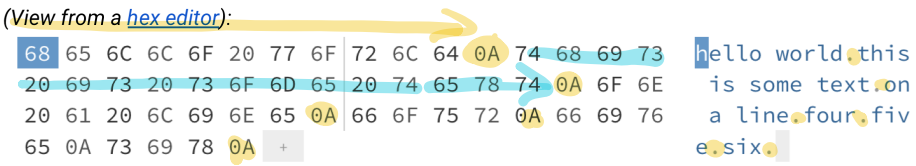
```
with open(DISK_PATH/"test.txt", "r") as f:
    while True:
        line = f.readline()
        if line == "":
            break

        print(line.strip())
```

=> readline() just scans the list of characters until it finds the newline

This is actually just a linear search of all the characters in the file!

(View from a hex editor):



So: can you jump to a specific line in a file?

=> Not with the tools that we have: don't know how far apart the newline characters are--need to read the file line by line until you get to the line you want
=> To learn how to do better, check out CS 300!

Similarly, if you wanted to delete a line, would need to read the whole file in, and then write again (without the line you want to delete)
=> Essentially, build the big array of characters all over again

Looking closer at the file....

test.txt

disk > test.txt

```
1  hello world
2  this is some text
3  on a line
4  four
5  five
6  six
7
```

If you wanted to delete a line, would need to read the whole file in, and then write again (without the line you want to delete)
=> Essentially, build the big array of characters all over again

(View from a hex editor):

Newlines.

```
68 65 6C 6C 6F 20 77 6F 72 6C 64 0A 74 68 69 73   hello world.this
20 69 73 20 73 6F 6D 65 20 74 65 78 74 0A 6F 6E    is some text.on
20 61 20 6C 69 6E 65 0A 66 6F 75 72 0A 66 69 76    a line.four.fiv
65 0A 73 69 78 0A                                 e.six.
```

What does readline() do?

```
with open(DISK_PATH/"test.txt", "r") as f:
    while True:
        line = f.readline()
        if line == "":
            break

        print(line.strip())
```

Just scans the list of characters until it finds the newline

File isn't an array of lines, actually an array of characters

So: can you jump to a specific line in a file?

Don't know how far apart the newline characters are => need to read the file line by line until you get to the line you want