Announcements
 - This is our last lecture
 - HW6: due on Monday
 - Regular office hours officially end this week
   => Some hours next week, but schedule will be different,
      see calendar for details

 - Look for announcement on grade reports

 - Final exam prep guide linked on lectures page
      => Look for announcement soon; review session TBA

Scenario 1: Online Listings for Room Rentals

Renting out rooms for a weekend convention

- Each listing contains the rental price for the weekend, the number of beds the unit has, the street address, and an optional photo. The site also stores the email address of the person who posted each listing (email addresses won't be displayed on the site)

- When a renter visits the site, they can search on either or <u>both</u> of the **number of beds** or the **price range**. The site displays those listings that match the search requirements.

- Listings get added to or removed from the site throughout the time that the site is up.

Questions

Which actions occur frequently, and thus need efficient run-time?

What data structure(s) do you propose? Include types of elements/keys/etc

Note: these problems are a bit harder than we would ask on the final—we are using more challenging problems here so we have more to discuss!

Scenario #1: A convention is coming to town, and local residents want to make some money by renting out all or part of their homes. In true do-it-yourself mode, they decide to build an online listings system from scratch. You've been hired to plan the data structures for the listings. Each listing contains the rental price for the weekend, the number of beds the unit has, the street address, and an optional photo. The site also stores the email address of the person who posted each listing, but the email addresses won't be displayed on the site. When a renter visits the site, they can search on either or both of the number of beds or the price range. The site displays those listings that match the search requirements. Listings get added to or removed from the site throughout the time that the site is up.

ACTIONS

DATA

Question 1: Which problem actions occur frequently, and thus need an efficient run-time?

↳ SEARCH BY NUM. BEDS AND/OR PRICE RANGE

ADD/REMOVE LESS FREQUENT ⇒ LESS IMPORTANT

Question 2: What data structure(s) do you propose for the listings? Be sure to include details such as the type of elements in lists, the types of keys/values in hashmaps, etc. Justify your data structure choice, in light of Question 1

   Proposals
   Two hashmaps
    - HashMap<Price, List<Listing>>
    - HashMap<#beds, List<...>>

NEED TO GET INTERSECTION

   Nested hashmap
   HashMap<Price<hashMap<#beds,List<listing>>>

```
=> Often okay to have multiple data structures to hold
   the same data to help us search
```

RANGE OF PRICES?? 150 - 225

100 - 170

First, Need a way to represent each listing (eg. object with price, beds, etc)
=> how to store efficiently to search?

LISTING
DEDS: 2
PRICE: 100
...

When discussing data structure choices: important to be specific on how data is used. Eg. if using hashmaps, what are keys/values?
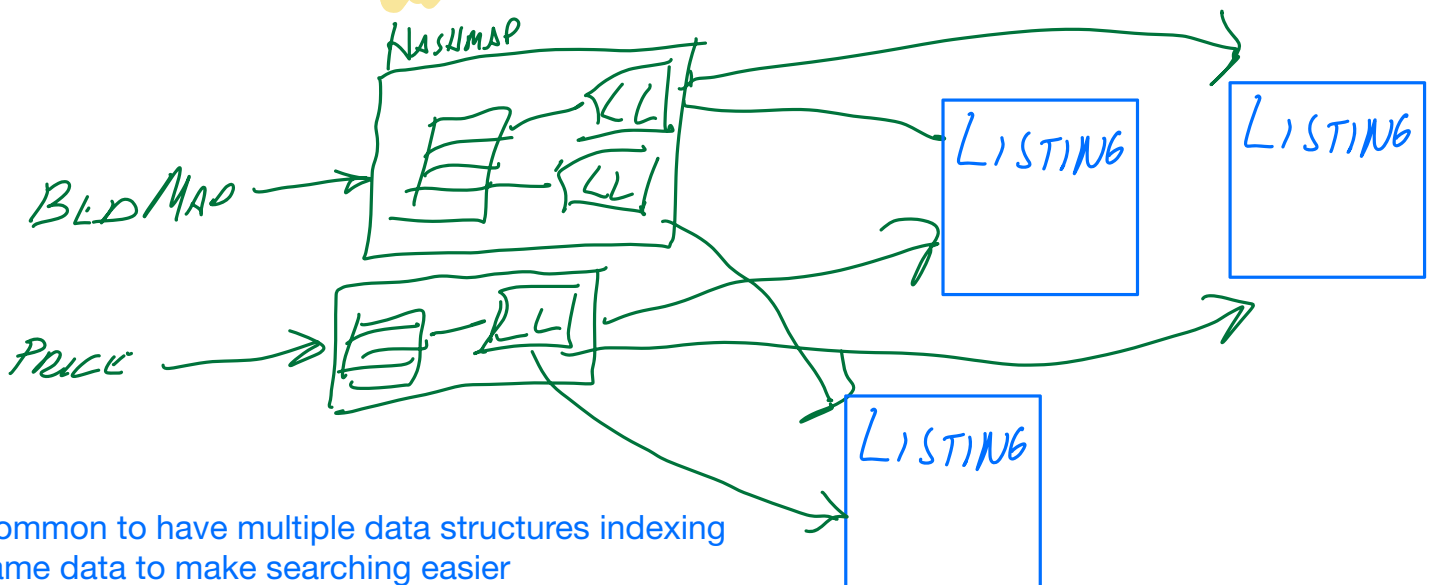
OPTIONS:

HASHMAP < PRICE , LIST OF LISTINGS >

HASH MAP < # BEDS , LIST OF LISTINGS >

ONE OR BOTH?

MAP< PRICE , MAP<# BEDS , LIST OF LISTINGS >>
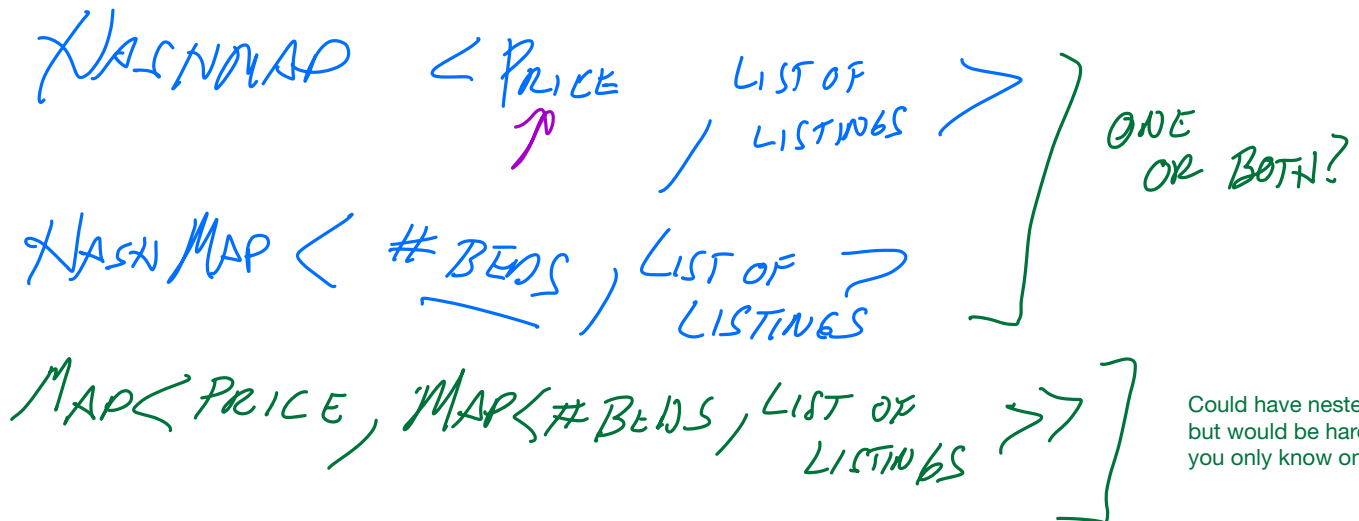
Could have nested hashmaps, but would be hard to search if you only know one parameter

On having two hash maps…

- HASH MAP < # BEDS, LIST< LISTING >>

HASH MAP < PRICE RANGE , LIST< LISTING >>

HASHMAP
<LL
<LL
BED MAP
PRICE
<LL
LISTING
LISTING
LISTING

Common to have multiple data structures indexing same data to make searching easier
 => Here, both maps refer to the same objects in memory, so this doesn't require extra space (need to keep both maps updated, though)

Q: Could you search on a range of prices with a HashMap?

=> Might be possible by making the key a pair or tuple (low,high), and then storing all listings in those ranges

$$HASH\ MAP < (LO,\ HI\ ),\ LIST < LISTING >>$$

$(150, 250)?$

$(100, 200)$	→ L → L → L
$(200, 300)$	→ L → L → ...
$(300, 400)$	→ □ → □ → ...
⋮	
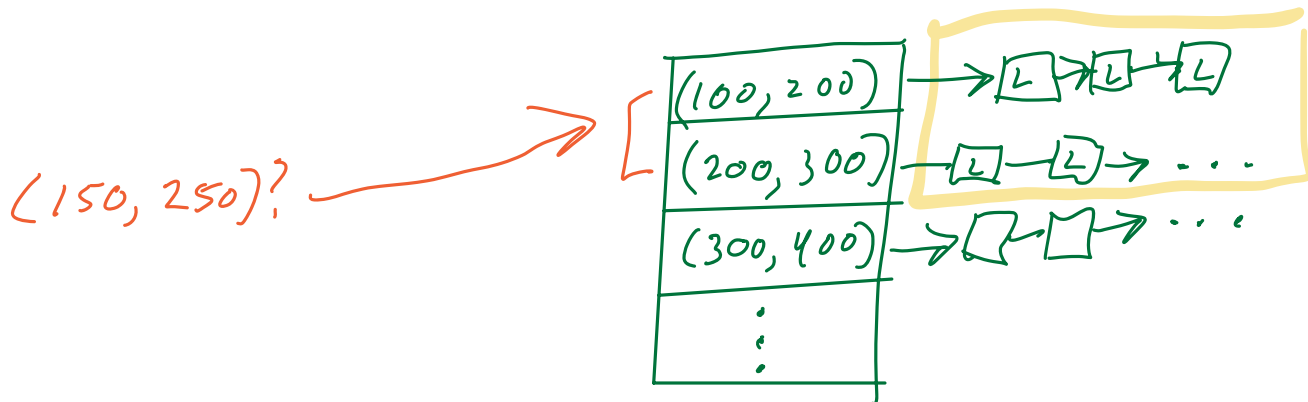
But what happens if we search for a price range that doesn't match the bounds in the map?

Example: want (150,250)
 => Would need to get all values in (100,200), (200,300), then iterate over them and pick the ones that match...
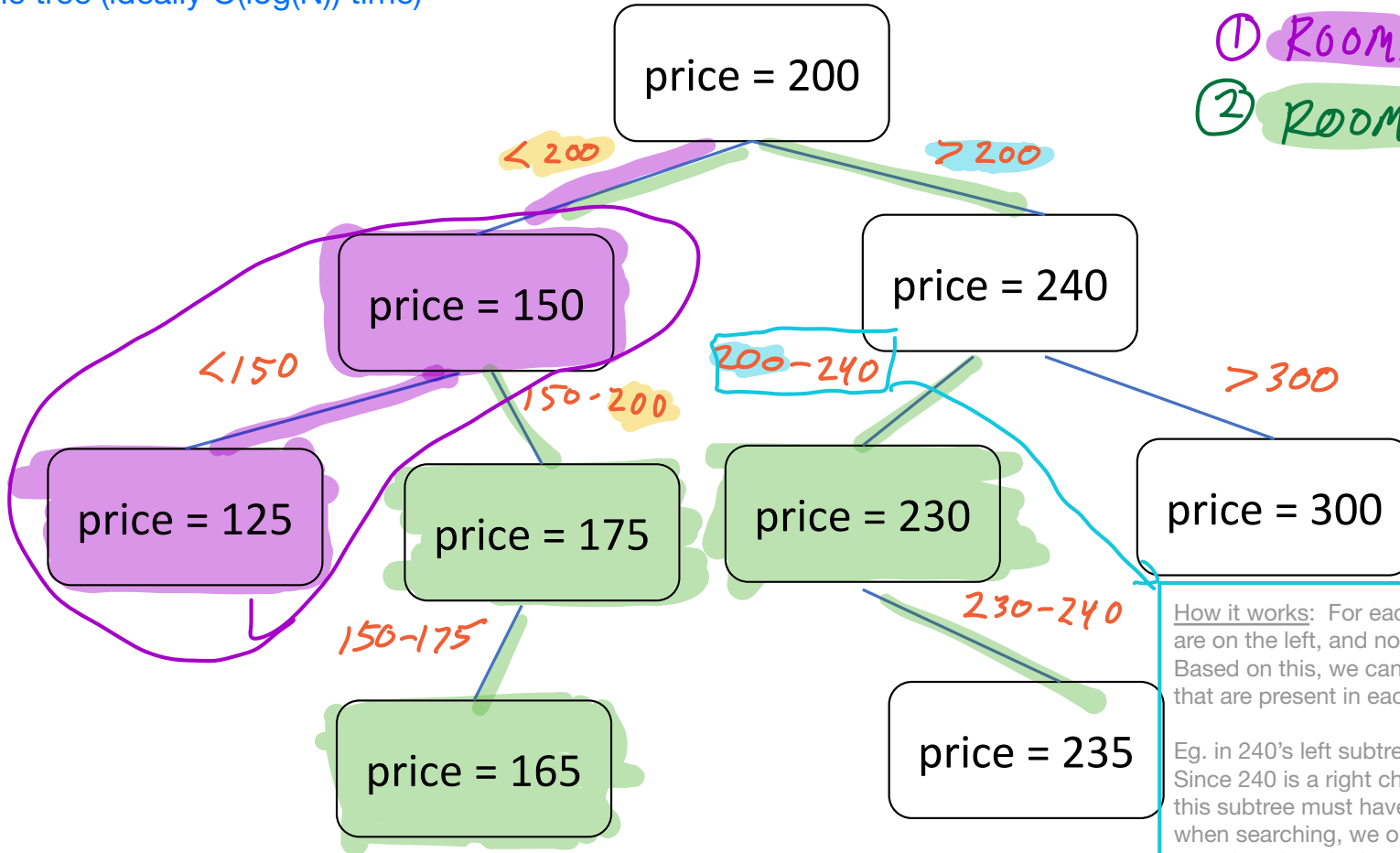
Can we do better???

How to search over different ranges of prices?
Could use a BST => ordering means we only need to traverse part of the tree (ideally O(log(N)) time)

EXAMPLES
① ROOMS UNDER 160
② ROOMS 160 - 230

price = 200

< 200   > 200

price = 150

< 150   150 - 200

200 - 240   > 300

price = 240

price = 125   price = 175

price = 230   price = 300

150 - 175   230 - 240

price = 165

price = 235

How it works: For each BST node, nodes with a lower value are on the left, and nodes with a higher value are on the right. Based on this, we can figure out the total range of numbers that are present in each subtree.

Eg. in 240's left subtree, all nodes must be <240. Since 240 is a right child of 200, they must also be >200, so this subtree must have values in the range 200-240. Thus, when searching, we only need to consider a subtree if it could contain values in the range we want to check.

Note: this is quite subtle and more than we would ask on the final.

Scenario 2: Making a Simple Web-based Document Editor

You've decided to create a web-based document editor (like a simplified Google Docs). You'll focus on four operations: adding text, deleting text, undoing edits, and searching for words in the document. The editor should allow documents to have basic styling, such as boldface words/phrases and section headings.

In particular, searching for words needs to be fast (because you imagine people writing large documents in your tool once it becomes popular).

Questions

What data do you need to manage for this problem?

What data structure(s) do you propose? Include types of elements/keys/etc

Need to store
  - **Documents**
  - **Edits/info to enable undo**

Two options for document
  - **One long string => not good for searching**
  - **List of words**

In general, how do we capture edits? What is an edit?

$$\overset{0}{\text{CS 200}} \quad \overset{1 \quad 15}{\text{WAS}} \quad \overset{2}{\text{HERE}} \rightarrow \overset{0}{\text{CS200}} \quad \overset{1}{\text{IS}} \quad \overset{2}{\text{HERE}}$$

EDIT: REPLACED WORD(1) WITH "WAS"

Common practice to store edits, not changed data structures
 => Functional programming vs. notation (remember immutable lists?)
 => Version control systems like git

This is an important in distributed systems, where computations are performed by many unreliable machines running in parallel
    => CS1380: Distributed Systems

```
<html>
  <p>
    This is a
    <b>bold</b>
    word, followed by
    <b>another one</b>
  </p>
</html>
```

=> Structured way to look at data
If you look hard at it...

Node class:
contents: tag-b
children: Node("BIG"), Node(bold)
previous: copy of the previous node in this spot

TREE!

```
<html>
  <p>
    This is a
    <b>BIG bold</b>
    word, followed by
    <b>another one</b>
  </p>
  <p>
    new para
  <p>
</html>
```
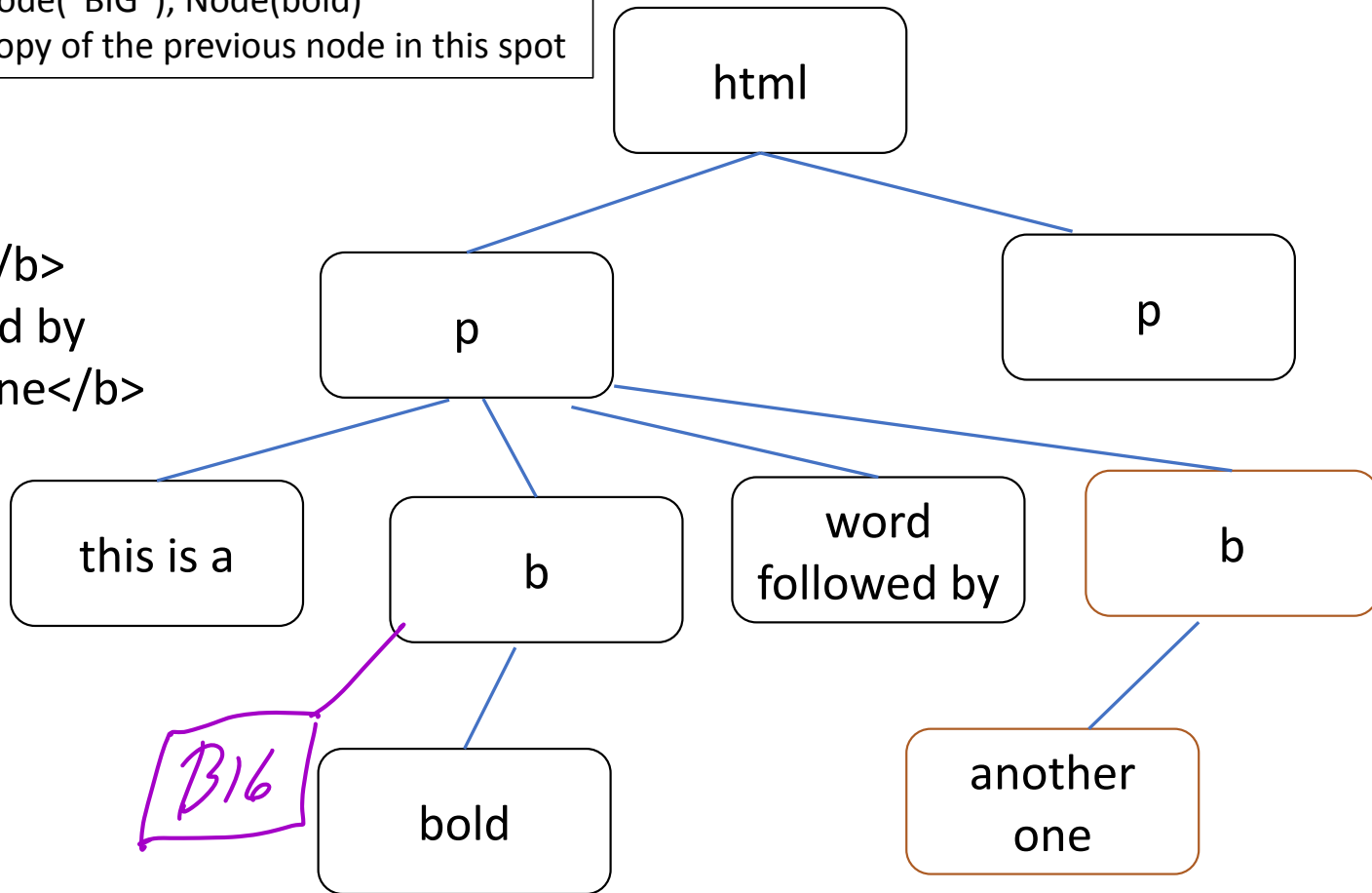
html

p

p

this is a

b

word followed by

b

BIG

bold

another one

Q: What was the point of each homework?
 - HW1: Learn Java
 - HW2: How to navigate a data structures (by learning how linked lists work)
 - HW3: idea of hashcodes, using arrays, hashmap structure
 - Decision tree: real-life use of trees and recursion; Intro to AI
 - HW4/GraphQuest: power of interfaces, how to traverse graphs
 - HW5: how hashing shows up in another context, how files work
 - HW6: how to use data structures to solve problems, optimize in different contexts

Things to think about as you leave CS200:

1. Runtime isn't everything. Sometimes a simpler data structure that
   will create fewer bugs is better than doing something more
   complicated

2. Code is just one part of writing software (tests, planning, docs,
analysis on how it works) => all of this is just as important, esp.
with ChatGPT

3. You will have bugs. This is normal, we all do it. Don't think of
bugs as something that says you don't belong in CS—it's just part of
what we do

4. Test as you go. As you write small bits of code, try to test it.
Write code, watch it break, do a test. Assignments have tried to walk
you through this => try to do it on your own, it will help!

5. Problems solve you: working on a problem changes you and how you
understand the world => goal is to improve your learning for future
problems. If you don't get something 100%, fine, get to 85-90% and
move on.

6. You belong here. Don't let yourself talk yourself out of it. If
you're enjoying CS, stick with it. Don't feel like you get it at this
point—I didn't feel like I understood some of this stuff until I
started teaching it

Thank you for a great semester!!!

Please stay in touch!