CODEPATH*ORG

Welcome to class!

Turn on your camera

Rename yourself to include your pod # at the beginning of your Zoom name -
ie 6 - Emily Garvey (she/her)

Answer the following question in the chat:
What is your favorite TV show | book | thing?

CODE PATH *ORG

Agenda

Check In	5 mins
Core Data Structures: Stacks and Queues	25 mins
Breakout Sessions	25 mins
BREAK	5 mins
Breakout Sessions	30 mins
Go Over Problem 1	25 mins
Wrap Up	5 mins

Announcements

- ❏ TA Office Hours

SE103 TA Office Hours Logistics

- **When are Office Hours held, and who hosts?**
  - Mondays from 10-11am PT / 1-3pm ET with Emily Crowl
  - Thursdays from 5-6pm PT / 8-9pm ET with Suresh Venkatesan
- **How do I join the meeting?**
  - You can find the zoom link in the “Schedule” page of the course portal
- **Are they mandatory?**
  - No, but we would love for you to join, even if you don’t have a specific question in mind
- **Are they recorded?**
  - Yes! The playlist link is also in the “Schedule” page, and the “Resources” page

CODE PATH *ORG

TA Office Hours Tips

- **What kinds of things will the TAs go over during Office Hours?**
  - TAs will usually prepare questions from the current or previous week -- including ones in the Session tabs, Resources tab, or from the assignments.
- **How can I make sure the TA goes over the topic I want them to cover?**
  - Ask them ahead of time! If you DM the TA before Office Hours, you can ensure they will prepare the question you want to discuss.
- **How can I make the most of Office Hours?**
  - Attend!! Even if you think you don't have a question, make it a routine to show up.
- **What if I can't attend Office Hours?**
  - You can post any questions you have in the #help slack channel. Our instructors, TAs, mentors, and fellow students all may be able to help. Posting code snippets, links, or specific questions are encouraged (just don't give away HackerRank answers before the due date!)

Core Data Structures: Stacks and Queues | 30 mins

CODE
PATH
*ORG

A quack guide to collections

https://turnoff.us/geek/java-collections/

CODE PATH *ORG

What is a Stack?

Stacks stores objects in a last in, first out (LIFO) fashion

Think of it in terms of stacks of plates

How does a stack work?

- The last thing added (pushed) is the first thing to be retrieved (popped)
- A stack is a sequence of items that are accessible at only one end of the sequence
- Operations that can be performed on a stack:
  - **`push`**: add an item to the top of the stack
  - **`pop`**: remove the item at the top of the stack
  - **`top/peek`**: get (but do not remove) the item at the top of the stack

Stack operations

```java
import java.util.Stack;

Stack<Object> stack = new Stack<>();
stack.add(1);
stack.add(2);
System.out.println(stack.pop());  // 2
System.out.println(stack.peek()); // 1
if (!stack.isEmpty()) {
   stack.pop(); // 1
}
```

```python
import collections

stack = collections.deque()
stack.append(1)
stack.append(2)
print(stack.pop()) # 2
print(stack[-1])   # 1
if stack:
  stack.pop()      # 1
```

CODE
PATH
*ORG

Postfix Notation

- Infix expression is the form AOB
  - A and B are numbers or also infix expression
  - O is operator (+, -, *, /)

- Postfix expression is the form ABO
  - A and B are numbers or also postfix expression
  - O is operator (+, -, *, /)

Expression notations

Infix notation

- A <op> B
- Examples
  - 3 + 5
  - 6 * 4 + 10 / 2

Postfix notation

- A B <op>
- Examples
  - 3 5 +
  - 6 4 * 10 2 / +

Expression notations

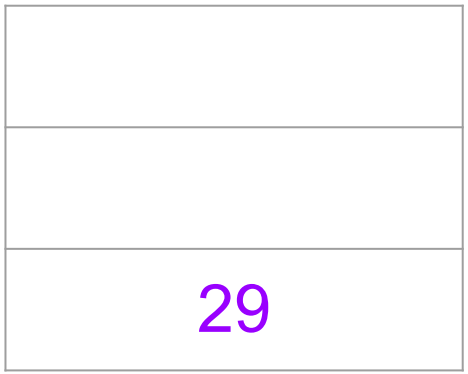
2
15
29

Postfix notation

- A B <op>
- Examples
  - 3 5 +
  - 6 4 * 10 2 / +

CODE PATH *ORG

Expression notations



Postfix notation

- A B <op>
- Examples
  - 3 5 +
  - 6 4 * 10 2 / +
- Advantage over infix
  - Parentheses never needed
  - Easier to evaluate

What is a Queue?

Queues store objects in a first in, first out (FIFO) fashion.

Think of it like waiting in line to cast your ballot.

CODE PATH *ORG

Queue operations

```java
import java.util.Queue;

Queue<Object> queue = new LinkedList<>();
queue.add(1);
queue.add(2);
queue.remove(); // 1
if (!queue.isEmpty()) {
    obj = queue.remove(); // 2
}
```

```python
import collections

queue = collections.deque()
queue.append(1)
queue.append(2)
queue.popleft() # 1
if queue:
  print(queue.popleft()) # 2
```

Queues: Key Takeaways

- Useful when ordering of the data matters as it preserves that ordering.
- Make sure you know the right classes and methods for your language.

Tips on what data structure to use

- Learning what data structure to use takes practice.
- When reviewing a data structure:
  - How does this compare to other data structures I know?
  - What does this data structure do well?
  - What does this data structure do poorly at?

- When you read a solution to a coding problem, spend time analyzing why a data structure was optimal for the problem.

Comparison

	finding max/min value	lookup	ordering by time
Heap			
Hash			
Stack			
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	**+**	**-**	**-**
Hash			
Stack			
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash			
Stack			
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	**-**	**+**	**-**
Stack			
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	-	+	-
Stack			
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	-	+	-
Stack	-	-	**LIFO**
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	-	+	-
Stack	-	-	LIFO
Queue			

Comparison

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	-	+	-
Stack	-	-	LIFO
Queue	**-**	**-**	**FIFO**

Comparison: the fine print

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	-	+	-
Stack	-	-	LIFO
Queue	-	-	FIFO

- There are types of hash sets/tables that keep track of:
  - relative ordering of entries (`TreeMap` and `TreeSet` in Java)
  - when items were added (`LinkedHashSet` in Java)
  - as of python 3.7, dictionaries maintain order of when items were added

In Class Walkthrough | 15 mins

CODE
PATH
*ORG

1021. Remove Outermost Parentheses

https://leetcode.com/problems/remove-outermost-parentheses/

A valid parentheses string is either:

- empty ""
- "(" + A + ")"
- or A + B

where A and B are valid parentheses strings, and + represents string concatenation.

For example:

"", "()", "(())()", and "(()(()))" are all **valid parentheses strings**.

CODE
PATH
*ORG

1021. Remove Outermost Parentheses

https://leetcode.com/problems/remove-outermost-parentheses/

A valid parentheses string *s* is primitive if it is nonempty, and there does not exist a way to split it into *s* = *A* + *B*, with A and B nonempty valid parentheses strings.

Given a valid parentheses string *s*, consider its primitive decomposition:

s = *P1* + *P2* + ... + *Pk*, where Pi are primitive valid parentheses strings.

Return *s* after removing the outermost parentheses of every primitive string in the primitive decomposition of *s*

Constraints:
1 <= s.length <= 105
s[i] is either '(' or ')'
s is a valid parentheses string

CODE
PATH
*ORG


THIS PROBLEM IS
TOO DAMN HARD
imgflip.com

1021. Remove Outermost Parentheses

https://leetcode.com/problems/remove-outermost-parentheses/

Example 1:

Input: *s* = "(()())(())"

Output: "()()()"

Explanation:
The input string is "(()())(())", with primitive decomposition "(()())" + "(())"
After removing outer parentheses of each part, this is "()()" + "()" = "()()()”

CODE
PATH
*ORG

1021. Remove Outermost Parentheses

https://leetcode.com/problems/remove-outermost-parentheses/

Example 2:

Input: s = "(()())(())(()(()))"

Output: "()()()()(())"

Explanation:

The input string is "(()())(())(()(()))", with primitive decomposition "(()())" + "(())" + "(()(()))".

After removing outer parentheses of each part, this is "()()" + "()" + "()(())" = "()()()()(())"

1021. Remove Outermost Parentheses

https://leetcode.com/problems/remove-outermost-parentheses/

Example 3:

Input: s = "()()"

Output: ""

Explanation:

The input string is "()()", with primitive decomposition "()" + "()"

After removing outer parentheses of each part, this is "" + "" = ""

CODE
PATH
*ORG

M-atch

Make use of a stack and a set

- Use a stack to operate the string
- Or… instead of using set to store the index to remove, we can directly use list slice to append the string to answer
- Skipping first and last element of the valid sub-parentheses requires some extra attention.

P-lan:

1. Use a stack to operate the string
2. Append “(“ when there is “(“ and pop “(“ there is meet “)”

P-lan:

3. We also need a set to store the indexes that need to remove

4. After pop, once the stack is empty we add the last value index and current value index into set.

5. Then we iterate the string again

6. Skip those index in set and concatenate others character to the answer.

I-mplement

```
def removeOuterParentheses(self, S: str) -> str:
        stack = []
        index remove=set()
        res=""
        for i, v in enumerate(S):
            if v == "(":
                stack.append(i)
            else:
                left_index = stack.pop()
                if not stack:
                    index remove.add(left_index)
                    index_remove.add(i)
        for i, v in enumerate(S):
            if i not in index_remove:
                res+=v
         return res
```

CODE
PATH
*ORG

R-eview:

- Trace through your code with an input to check for the expected output.
- Catch possible edge cases and off-by-one errors.

E-valuate

Time complexity

- O(2n), since we iterate the list twice so the total cost will be roughly

Space complexity

- O(n), since we need to store items in stack

Breakout Sessions | 60 mins

CODE
PATH
*ORG

Problem #1: Brick Wall

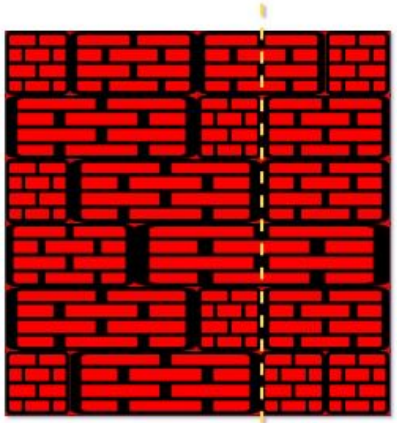
Original Problem on Leetcode: Brick Wall

There is a rectangular brick wall in front of you with `n` rows of bricks. The `ith` row has some number of bricks each of the same height (i.e., one unit) but they can be of different widths. The total width of each row is the same.

Draw a vertical line from the top to the bottom and cross the least bricks. If your line goes through the edge of a brick, then the brick is not considered as crossed. You cannot draw a line just along one of the two vertical edges of the wall, in which case the line will obviously cross no bricks.

Given the 2D array `wall` that contains the information about the wall, return the minimum number of crossed bricks after drawing such a vertical line.

Example:



```
Input: wall = [[1,2,2,1],[3,1,2],[1,3,2],[2,4],[3,1,2],[1,3,1,1]]
Output: 2
```

CODE PATH *ORG

Activity | Breakout Groups

60 MINS

Directions

1. IceBreaker: You get a million dollars but you must legally change your first name to Buttergunt for the rest of your life. Do you do it? Why?
2. Work together through the UMPIRE steps of the problems in the Course Portal under **Week 2, Session 2 Brick Wall**
3. If your pod needs help, post a message on the slack channel and tag **@se103-tas**

Reminders

Don't forget to turn on your cameras!

Take a 5 min break sometime during your breakout session!

CODE PATH *ORG

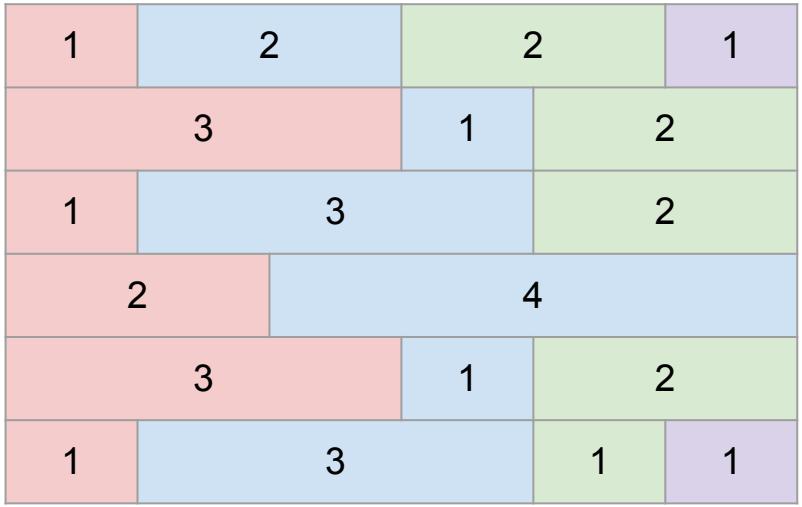
Breakout Problem Review | 15 mins

Don’t peek at these until I present them!

CODE
PATH
*ORG

U-nderstand

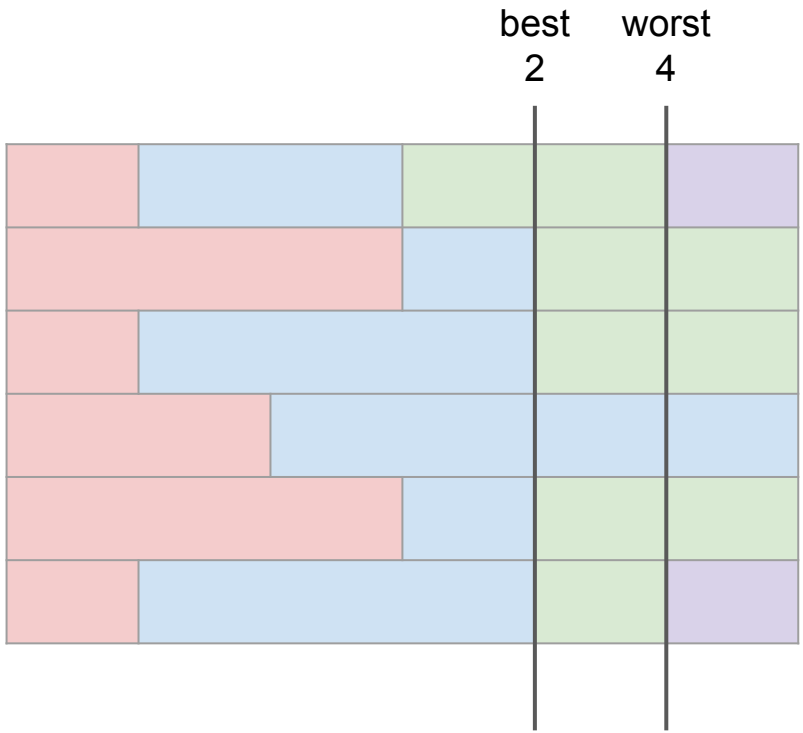
```
Input: wall = [
  [1,2,2,1],
  [3,1,2],
  [1,3,2],
  [2,4],
  [3,1,2],
  [1,3,1,1]
]

Output: 2
```



CODE
PATH
*ORG

U-nderstand

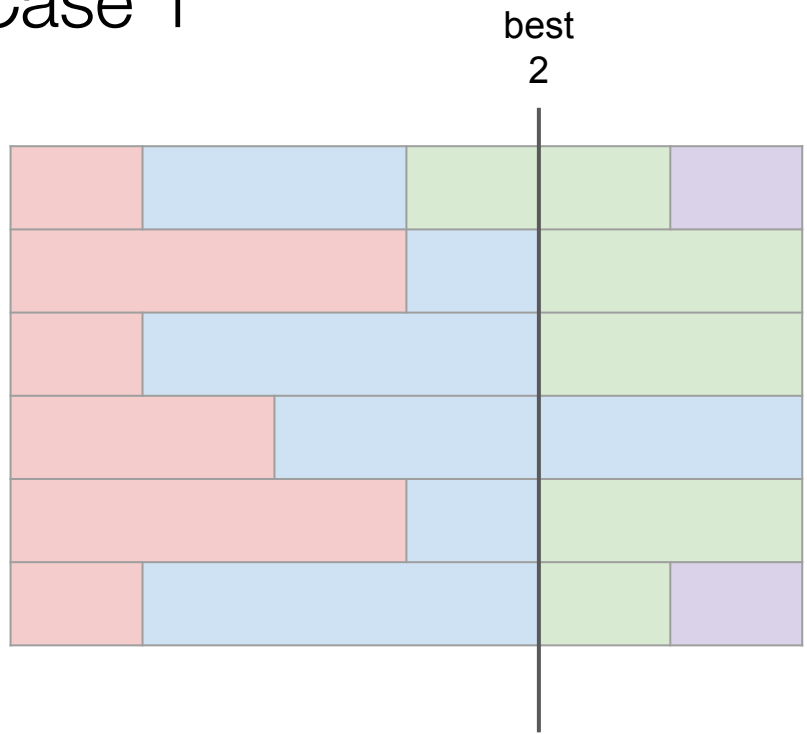
```
Input: wall = [
  [1,2,2,1],
  [3,1,2],
  [1,3,2],
  [2,4],
  [3,1,2],
  [1,3,1,1]
]

Output: 2
```



CODE
PATH
*ORG

U-nderstand: Test Case 1

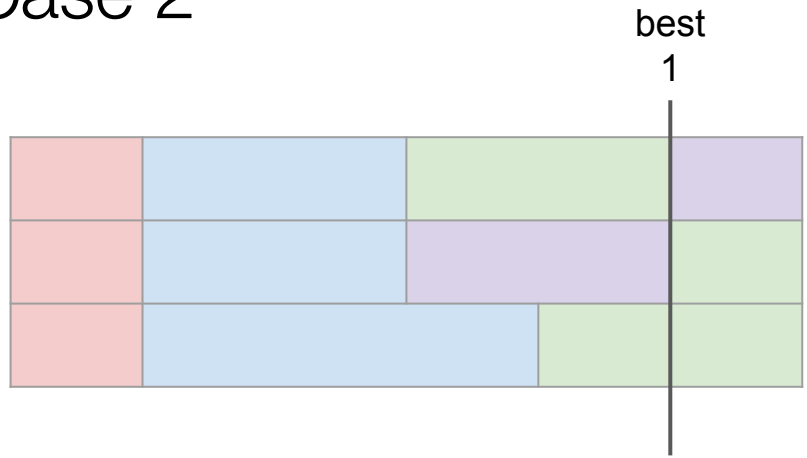
```
Input: wall = [
  [1,2,2,1],
  [3,1,2],
  [1,3,2],
  [2,4],
  [3,1,2],
  [1,3,1,1]
]

Output: 2
```



CODE
PATH
*ORG

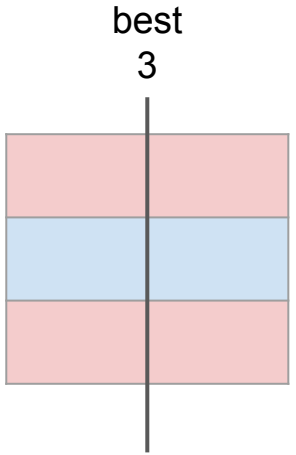
U-nderstand: Test Case 2

```
Input: wall = [
  [1,2,2,1],
  [1,2,2,1],
  [1,3,2],
]

Output: 1
```



U-nderstand: Test Case 3 (edge case)

```
Input: wall = [
  [2],
  [2],
  [2],
]

Output: 3
```

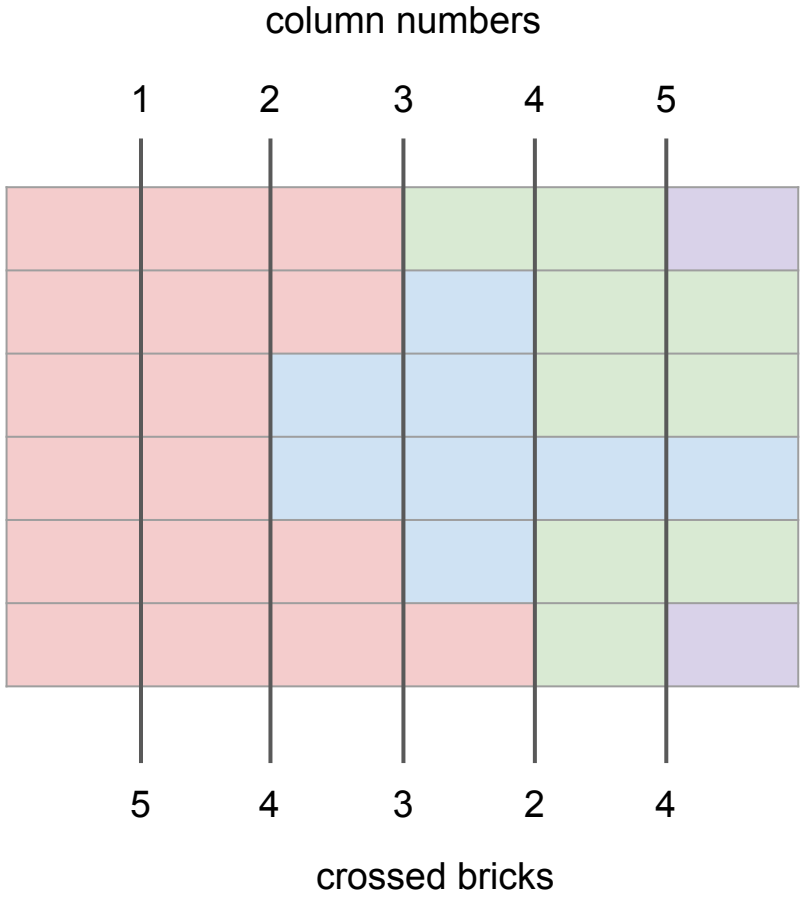


CODE
PATH
*ORG

U-nderstand: Test Case 4 (edge case)

```
Input: wall = []
Output: 0
```

M-atch

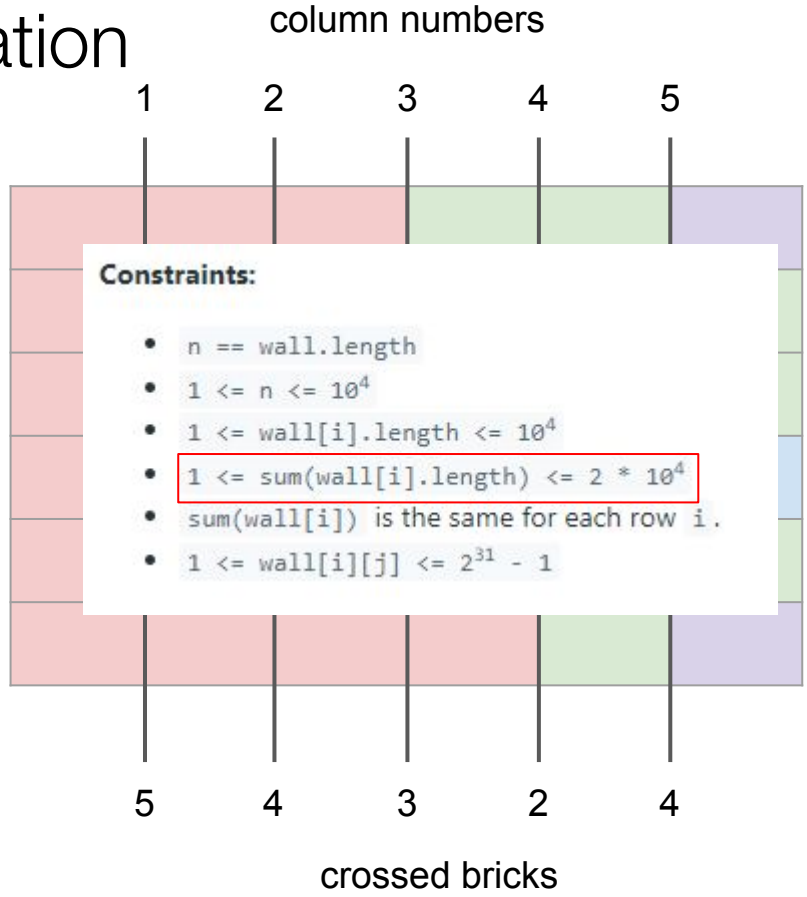


1. Build a representation of the wall.
2. Find the column with the fewest bricks.

M-atch: Representation

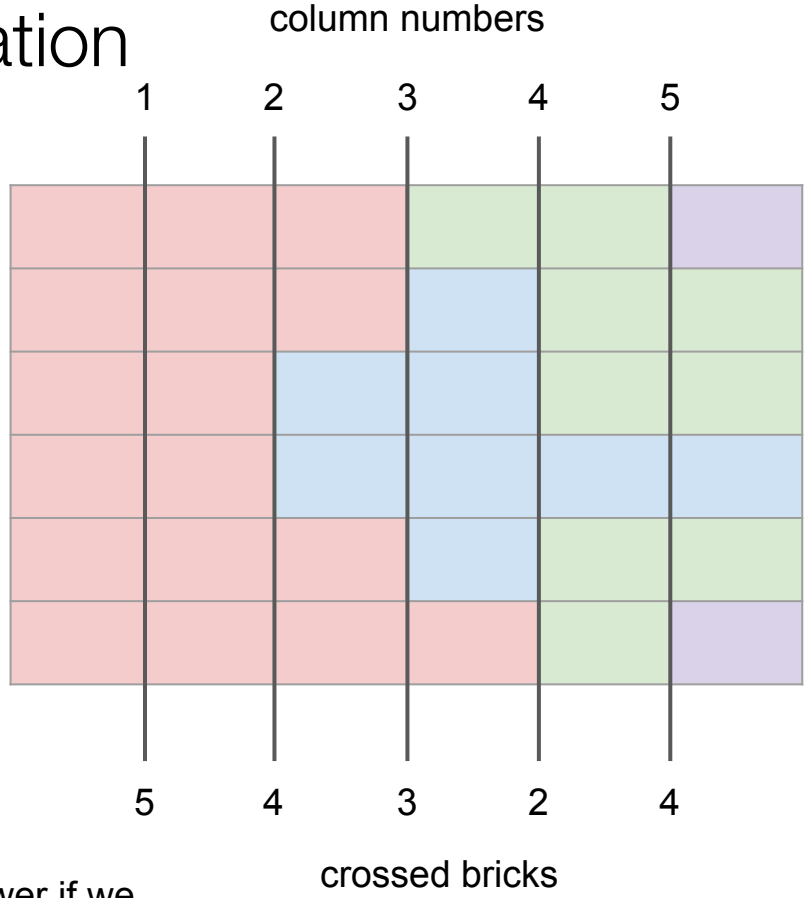
We'll need to keep count of a number of bricks spanning each column.

How about an array?



M-atch: Representation

```
Input: wall = [
  [3,2,1],
  [3,1,2],
  [2,2,2],
  [2,4],
  [3,1,2],
  [4,1,1]
]

Output: 2
```



We'll need to keep count of a number of bricks spanning each column.

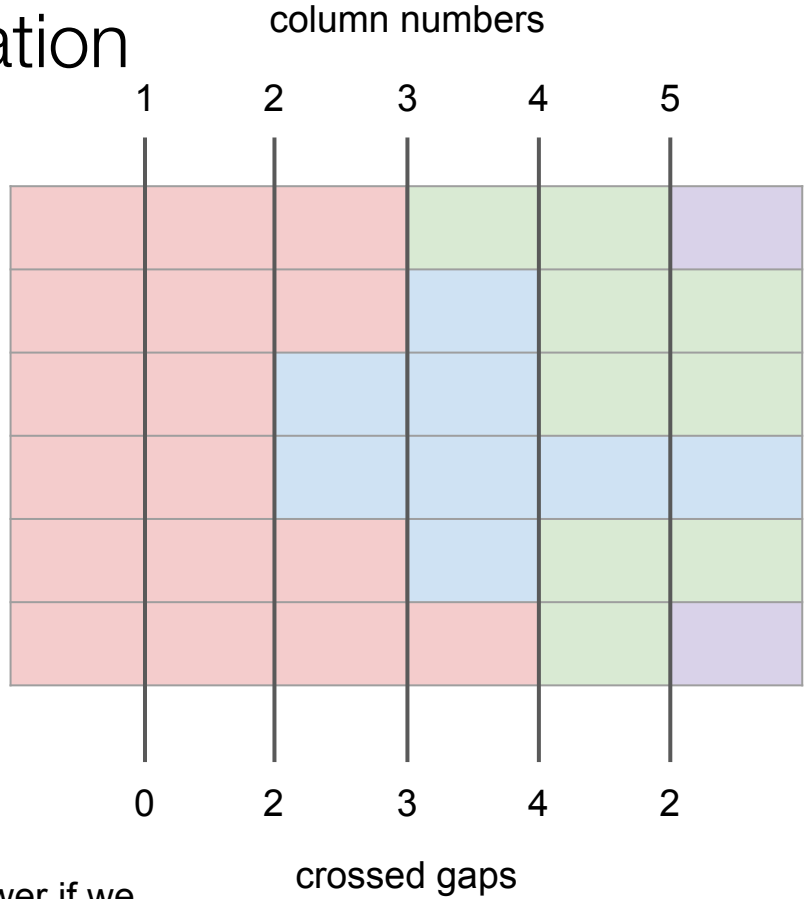
How about an array? No

Consider a hash table instead.

That's still a lot of entries. Maybe there will be fewer if we keep track of gaps instead.

M-atch: Representation

```
Input: wall = [
  [3,2,1],
  [3,1,2],
  [2,2,2],
  [2,4],
  [3,1,2],
  [4,1,1]
]

Output: 2
```



We'll need to keep count of a number of bricks spanning each column.

How about an array? No

Consider a hash table instead.

That's still a lot of entries. Maybe there will be fewer if we keep track of gaps instead.

CODE PATH *ORG

P-lan

1. Build a hash table to keep track of gaps per column.
2. For each row in rows:
    a. Set x to 0
    b. For each brick in the row:
        i. x = x + brick.width
        ii. Increment gaps[x]
3. Find the maximum number of gaps (max value in gaps).
4. Return {number of rows} - {max number of gaps} to get fewest bricks crossed.

I-mplementation

```java
import java.util.*;

public int leastBricks(List<List<Integer>> wall) {
    Map<Integer, Integer> map = new HashMap<>();

    for (List<Integer> row : wall) {
        int x = 0;
        for (int i = 0; i < row.size() - 1; i++) {
            x += row.get(i);
            int gaps = map.getOrDefault(x, 0) + 1;
            map.put(x, gaps);
        }
    }
    return wall.size() - Collections.max(map.values());
}
```

```python
import collections

def leastBricks(self, wall):
    gaps = defaultdict(int)

    for bricks in wall:
        x = 0
        for brick in bricks[:-1]:
            x += brick
            gaps[x] += 1

    maxGaps = max(gaps.values())
    return len(wall) - maxGaps
```

CODE
PATH
*ORG

R-eview

- Test case 1: pass
- Test case 2: pass
- Test case 3: fail
- Test case 4: fail

Test case 3

```
Input: wall = [
  [2],
  [2],
  [2],
]

Output: 3
```

Test case 4

```
Input: wall = [ ]

Output: 0
```

CODE
PATH
*ORG

I-mplementation (corrected)

```java
public int leastBricks(List<List<Integer>> wall) {
    Map<Integer, Integer> map = new HashMap<>();

    for (List<Integer> row : wall) {
        int x = 0;
        for (int i = 0; i < row.size() - 1; i++) {
            x += row.get(i);
            int gaps = map.getOrDefault(x, 0) + 1;
            map.put(x, gaps);
        }
    }
    if (map.size() > 0) {
        return wall.size() -
           Collections.max(map.values());
    } else {
        return wall.size();
    }
}
```

```python
def leastBricks(self, wall):
    gaps = defaultdict(int)

    for bricks in wall:
        x = 0
        for brick in bricks[:-1]:
            x += brick
            gaps[x] += 1

    maxGaps = max(gaps.values()) if gaps else 0
    return len(wall) - maxGaps
```

CODE PATH *ORG

R-eview (2nd time)

- Test case 1: pass
- Test case 2: pass
- Test case 3: pass
- Test case 4: pass

E-valuate: time

1. Build a hash table to keep track of gaps per column.
2. For each row in rows:
    a. Set x to 0
    b. For each brick in the row:
        i. x = x + brick.width
        ii. Increment gaps[x]
3. Find the maximum number of gaps (max value in gaps).
4. Return {number of rows} - {max number of gaps} to get fewest bricks crossed.

O(n), where n is the number of bricks.

E-valuate: space

1. Build a hash table to keep track of gaps per column.
2. For each row in rows:
    a. Set x to 0
    b. For each brick in the row:
        i. x = x + brick.width
        ii. Increment gaps[x]
3. Find the maximum number of gaps (max value in gaps).
4. Return {number of rows} - {max number of gaps} to get fewest bricks crossed.

O(g), where g is the number of columns with gaps

Wrap Up | 10 mins

CODE
PATH
*ORG

Activity | Exit Ticket

3 MINS

Answer the 3 multiple choice in the Zoom Poll.

Exit tickets are a great way for us to check your understanding of the topics we discussed in class and identify for you what you should review after class.

Exit Ticket: Question #1

What are **heaps** best for?

a) Keeping track of the maximum or minimum value
b) Quickly checking whether they contain a specific value
c) Retrieving values based on when they were added

Exit Ticket: Question #2

What are **hash tables** best for?

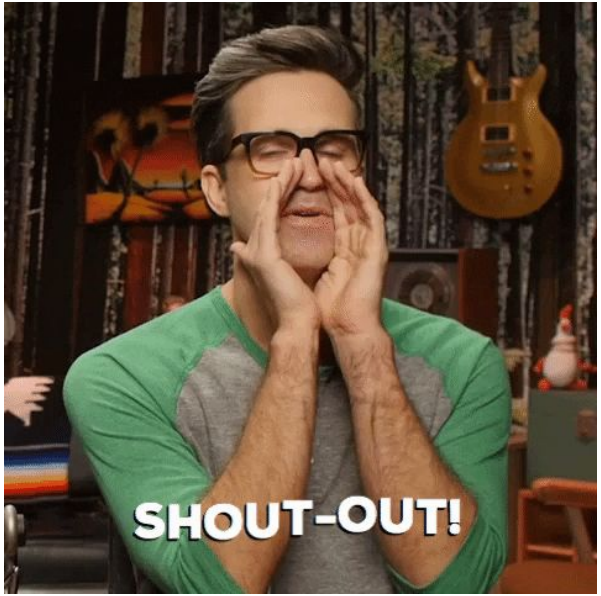
a) Keeping track of the maximum or minimum value
b) Quickly checking whether they contain a specific value
c) Retrieving values based on when they were added

Exit Ticket: Question #2

What are **stacks and queues** best for?

a) Keeping track of the maximum or minimum value
b) Quickly checking whether they contain a specific value
c) Retrieving values based on when they were added

Shout Outs!



Take a moment to shoutout someone today who helped you out.

Alternatively, drop in the chat something new that you were excited to learn about today!

CODE PATH *ORG

HackerRank Reminders

- You must use the **same** email you provided CodePath during application for your submission to be tracked
- HRs are due the day before Session #1 of the following week
  - Wed/Sat class: Tuesdays at 11:59pm PDT
- You are allowed 2 missing/submitted late HRs, no questions asked

At CodePath we collect your scores as a way to track your progress and provide you feedback throughout the program. You are not graded in this course, these assignments are made solely for practice only.

All information is on the **assignment tab** on the course portal.

CODE PATH *ORG

Before you Leave |

- ❏ Complete the **Session Survey** [5 min]
- ❏ Next session is Wed 09/28 5pm pst
- ❏ Complete your HackerRank assessment by **1 day before the next session** at 11:59pm PDT

Appendix

Comparison: the fine print

	finding max/min value	lookup	ordering by time
Heap	+	-	-
Hash	-	+	-
Stack	-	-	+ (LIFO)
Queue	-	-	+ (FIFO)