Strings and Arrays

Session 1

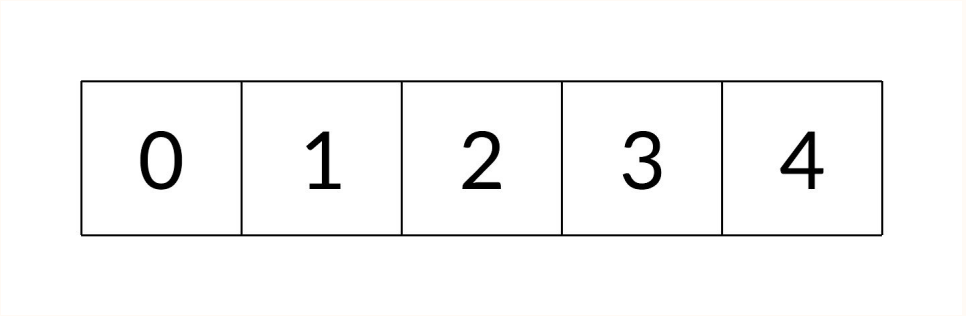
Overview

1. Arrays Overview
2. Strings Overview
3. Walkthrough
4. In class exercises

Arrays

Arrays

Data structure that holds a fixed number of objects.



Arrays

Strengths

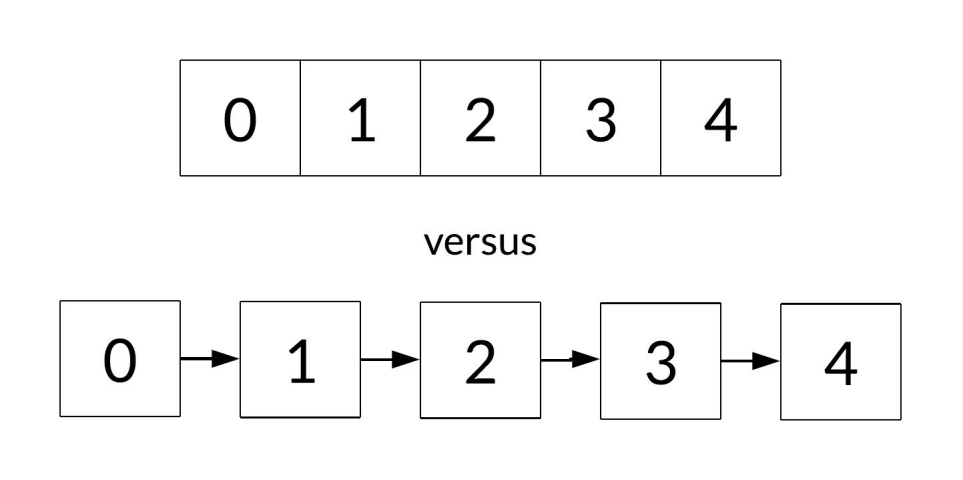
- Quick index based lookups
- Amortized quick insertion at the end of the list (dynamically sized arrays)

Weaknesses

- Fixed size (in certain languages)
- Inefficient deletion and insertions in the middle of the array

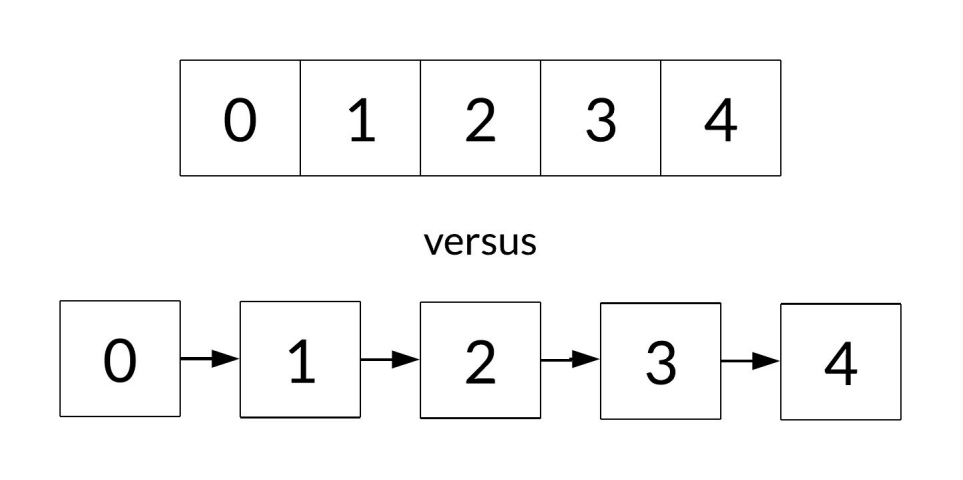
Arrays vs Linked Lists

1. What are some advantages of an array over a linked list?



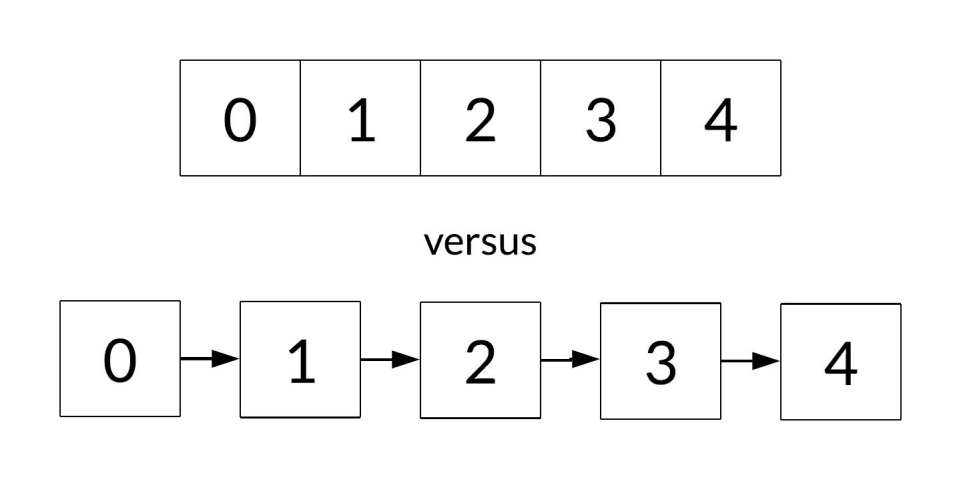
Arrays vs Linked Lists

1. What are some advantages of an array over a linked list?
    a. Contiguous memory usage
    b. Fast lookup



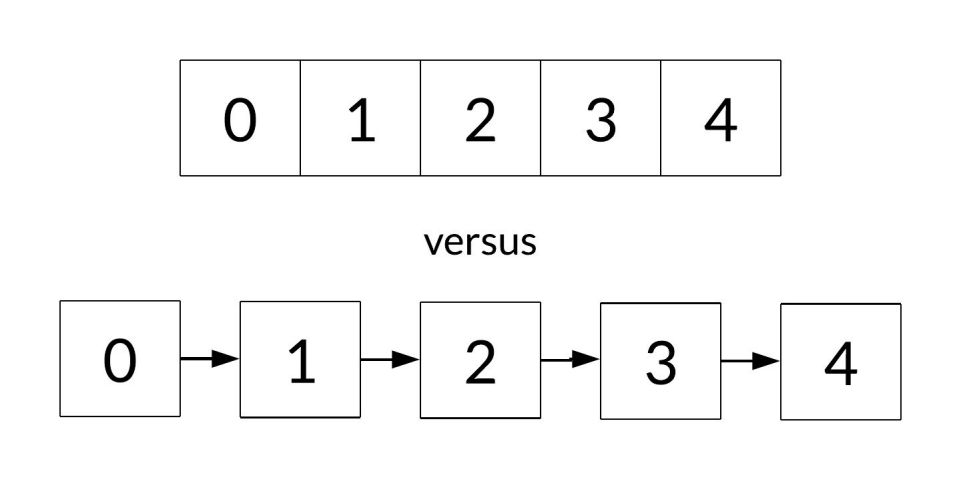
Arrays vs Linked Lists

1. What are some advantages of an array over a linked list?
    a. Contiguous memory usage
    b. Fast lookup
2. What are some advantages of a linked list over an array?



Arrays vs Linked Lists

1. What are some advantages of an array over a linked list?
    a. Contiguous memory usage
    b. Fast lookup
2. What are some advantages of a linked list over an array?
    a. Efficient for insert/ delete in the middle
    b. Not a fixed size



Array Interview Questions

- Common **array operations**
  - Indexing, appending, inserting, removing an element, getting the length/element
  - Reversing an array
  - Getting a subarray
  - Sorting an array
- **2D arrays** come up quite frequently
  - Video walkthrough of a 2D matrix question in the resources tab
- **Common traversals**: Binary Search, reverse order, matrix traversal

These should be muscle memory!

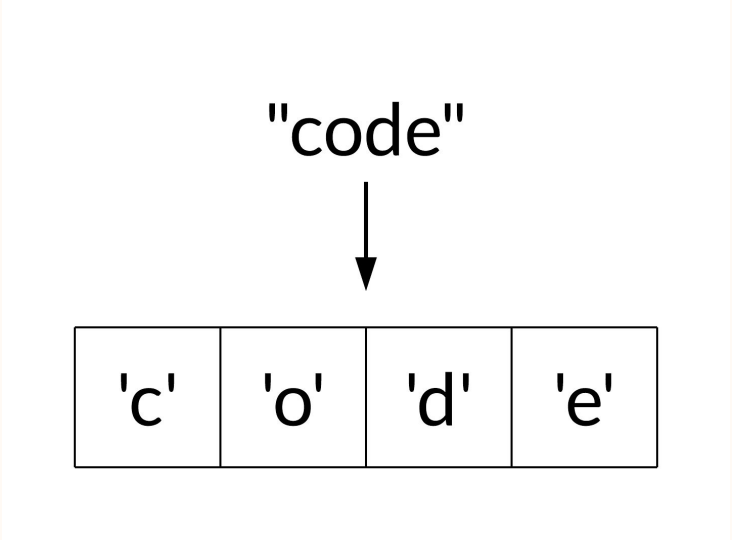
Common Array Mistakes

- Getting runtime complexities wrong
  - *search*, *remove*, *insert*
- Using fixed/dynamic size arrays incorrectly
- Using arrays to keep track of values
  - Use a set
- Off-by-one errors, wrong indexing
  - Especially for matrices

Strings

Strings

Special kind of array, one that only contains characters.



String Interview Questions

- Know your string operations (review the string library)
  - Indexing, appending, inserting, removing a character, getting the length
  - Sorting a string
  - Getting/finding a substring
  - Converting a character into an int (ascii value)
  - Splitting a string based on a delimiter
- Off-by-one errors
- Whenever you manipulate a string, a new copy of the string is created
  - In Java, you can use StringBuffer
  - In Python, you can convert a string to a list of chars

Practice! **These should be muscle memory!**

Sliding Window

Sliding Window

- "**Slide**" a static/dynamically sized window along the string/array
- Use a **variable/auxilliary data structure** to keep track of values (e.g. `maxSeenSoFar`)
- Can reduce brute force runtime to O(n) time
- A few different ways to utilize two pointers
    - Have both pointers start at the beginning
    - Have a pointer start at the beginning, another one start at the end
- Common in both array and string questions

Guide: https://guides.codepath.com/compsci/Two-pointer

Sliding Window Components

1. Two pointers (one to mark the start and end of the window)
2. While loop that keeps increasing the window by incrementing the end pointer by 1
3. Shrink the window by incrementing the start pointer when some condition is violated (this will depend on the problem).
    a. Use a counter or hash-map to help identify when the window is invalid
4. Update the current maximum window size, minimum window size, number of windows, etc. This will be the return value of the function.

Guide: https://guides.codepath.com/compsci/Two-pointer

Sliding Window

Keywords: contiguous, continuous, sequence, subarray, substring, min, max, longest, shortest

Guide with walkthrough: https://guides.codepath.com/compsci/Two-pointer

Walkthrough

Longest Substring with At Most Two Distinct Characters

Given a string **s** , find the length of the longest substring **t** that contains **at most** 2 distinct characters.

Longest Substring with At Most Two Distinct Characters

Given a string **s** , find the length of the longest substring **t** that contains **at most** 2 distinct characters.

Example 1:

```
Input: "eceba"
Output: 3
Explanation: t is "ece" which its length is 3.
```

Longest Substring with At Most Two Distinct Characters

Given a string **s** , find the length of the longest substring **t** that contains **at most** 2 distinct characters.

Example 1:

```
Input: "eceba"
Output: 3
Explanation: t is "ece" which its length is 3.
```

Example 2:

```
Input: "ccaabbb"
Output: 5
Explanation: t is "aabbb" which its length is 5.
```

Understand the problem

What would the output be for these inputs?

Input: `‘aabbbbcc’`

Understand the problem

What would the output be for these inputs?

Input: ‘aabbbbcc’

Output: 6 (either ‘aabbbbcc’ or ‘aabbbbcc’ would work)

Understand the problem

What would the output be for these inputs?

Input: `‘aabbbbcc’`

Output: 6 (either `‘aabbbbcc’` or `‘aabbbbcc’` would work)

Input: `"ababcbcbabbaadef"`

Understand the problem

What would the output be for these inputs?

Input: `'aabbbbcc'`

Output: 6 (either `'aabbbbcc'` or `'aabbbbcc'` would work)

Input: `'ababcbcbabbaade'`

Output: 6 (`'ababcbcbabbaadef'` would be the only answer)

Solving the problem: Brute force

Generate all possible substrings and count the number of unique characters for each substring

Solving the problem: Brute force

Generate all possible substrings and count the number of unique characters for each substring

```
def longest_substring_two_distinct(s):
    max_length = 0
    for start in range(len(s)):
        for end in range(start, len(s)):
            # Track the number of distinct characters in this substring
            distinct_characters = set()
            for char in s:
                distinct_characters.add(char)
            # Substring is a possible candidate if 2 or fewer distinct characters
            if len(distinct_characters) <= 2:
                max_length = max(max_length, end - start)
    return max_length
```

Solving the problem: Brute force

```
def longest_substring_two_distinct(s):
    max_length = 0
    for start in range(len(s)):
        for end in range(start, len(s)):
            # Track the number of distinct characters in this substring
            distinct_characters = set()
            for char in s:
                distinct_characters.add(char)
            # Substring is a possible candidate if 2 or fewer distinct characters
            if len(distinct_characters) <= 2:
                max_length = max(max_length, end - start)
    return max_length
```

Time complexity?

Solving the problem: Brute force

```
def longest_substring_two_distinct(s):
    max_length = 0
    for start in range(len(s)):
        for end in range(start, len(s)):
            # Track the number of distinct characters in this substring
            distinct_characters = set()
            for char in s:
                distinct_characters.add(char)
            # Substring is a possible candidate if 2 or fewer distinct characters
            if len(distinct_characters) <= 2:
                max_length = max(max_length, end - start)
    return max_length
```

Time complexity?

N^2

Solving the problem: Brute force

```
def longest_substring_two_distinct(s):
    max_length = 0
    for start in range(len(s)):
        for end in range(start, len(s)):
            # Track the number of distinct characters in this substring
            distinct_characters = set()
            for char in s:
                distinct_characters.add(char)
            # Substring is a possible candidate if 2 or fewer distinct characters
            if len(distinct_characters) <= 2:
                max_length = max(max_length, end - start)
    return max_length
```

Space Complexity?

Solving the problem: Brute force

```
def longest_substring_two_distinct(s):
    max_length = 0
    for start in range(len(s)):
        for end in range(start, len(s)):
            # Track the number of distinct characters in this substring
            distinct_characters = set()
            for char in s:
                distinct_characters.add(char)
            # Substring is a possible candidate if 2 or fewer distinct characters
            if len(distinct_characters) <= 2:
                max_length = max(max_length, end - start)
    return max_length
```

Space Complexity?

O(1)

How can this be improved?

Two Pointer/ Sliding Window approach seems like a good candidate to speed up run time!

Sliding window

Let’s try to work with this string `‘ccacbbabba’`:

Sliding window

Let’s try to work with this string `‘ccacbbabba’`:

6 would be the answer (`‘ccac`bbabba`c’`)

Sliding window

Let's try to work with this string `'ccacbbabba'`:

6 would be the answer (`'ccacbbabbac'`)

Remember: we always want to be evaluating valid windows (substrings with 2 characters)

Sliding window

start

c c a c b b a b b a

end

Longest valid substring so far: ‘c’

Sliding window

start

c c a c b b a b b a

end

Longest valid substring so far: ~~‘c’~~ ‘cc’

Sliding window

start

c c a c b b a b b a

end

Longest valid substring so far: ~~‘ee’~~ ‘cca’

Sliding window

start

c c a c b b a b b a

end

Longest valid substring so far: ~~‘cca’~~ ‘ccac’

Sliding window

start

c c a c b b a b b a

end

Longest valid substring so far: ‘ccac’

‘ccacb’ is no longer a valid substring

Sliding window

start

c c a c b b a b b a

end

Longest valid substring so far: ‘ccac’

We need to shrink the window by moving the start pointer to the right until it’s a valid window again.

Sliding window

~~start~~ start

c c a c b b a b b a

end

Longest substring so far: ‘ccac’

Sliding window

start

c c a c b b a b b a

end

Longest substring so far: `'ccac'`

Sliding window

start

c c a c b b a b b a

end

Another invalid substring, where do we move the start pointer?

Sliding window

start

c c a c b b a b b a

end

Another invalid substring, where do we move the start pointer?

We move the start pointer one position to the right.

Sliding window

~~start~~ start

c c a c b b a b b a

end

Longest substring so far: ‘ccac’

Sliding window

start

c c a c b b a b b a

end

Longest substring so far: ~~‘ccac’~~ ‘bbabb’

Sliding window

start

c c a c b b a b b a

end

Longest substring so far: ‘bbabba’

Sliding window plan

- If the current window only has 2 distinct characters, we **grow** the window by incrementing the end pointer
- If the current window is **no longer valid** (has 3 distinct characters), we **shrink** the window as little as possible until it the window is valid
    - Keep a mapping of characters in the window to their counts within the window to track when we can stop moving the start pointer to the right.
- Update the maximum substring length along the way

Pseudocode

```
char_to_counts = {} # dictionary that maps characters to its counts within the window

while end pointer isn't past the end of the string:
      Update end character count in char_to_counts
      while char_to_counts has more than 2 characters: # window is no longer valid
            decrement start character's count
            increment start index by 1
            remove entry from char_to_counts if count is 0 # character will no longer be in the window
      update the max length
      increment end pointer
```

Time / Space complexity

```
def longest_substring_two_distinct(s):
    char_counts = defaultdict(int)
    start, end, max_len = 0, 0, 0

    while end < len(s):
        # Fetch the newest character in the substring and update its count
        char_counts[s[end]] += 1

        # Too many distinct characters in the substring, we need to shrink the window
        while len(char_counts) > 2:
            start_char = s[start]
            char_counts[start_char] -= 1
            start += 1
            if char_counts[start_char] == 0:
                del char_counts[start_char]
        max_len = max(max_len, end - start + 1)
        end += 1
    return max_len
```

Time complexity?

Time / Space complexity

```
def longest_substring_two_distinct(s):
    char_counts = defaultdict(int)
    start, end, max_len = 0, 0, 0

    while end < len(s):
        # Fetch the newest character in the substring and update its count
        char_counts[s[end]] += 1

        # Too many distinct characters in the substring, we need to shrink the window
        while len(char_counts) > 2:
            start_char = s[start]
            char_counts[start_char] -= 1
            start += 1
            if char_counts[start_char] == 0:
                del char_counts[start_char]
        max_len = max(max_len, end - start + 1)
        end += 1
    return max_len
```

Time complexity?

O(N)

Time / Space complexity

```
def longest_substring_two_distinct(s):
    char_counts = defaultdict(int)
    start, end, max_len = 0, 0, 0

    while end < len(s):
        # Fetch the newest character in the substring and update its count
        char_counts[s[end]] += 1

        # Too many distinct characters in the substring, we need to shrink the window
        while len(char_counts) > 2:
            start_char = s[start]
            char_counts[start_char] -= 1
            start += 1
            if char_counts[start_char] == 0:
                del char_counts[start_char]
        max_len = max(max_len, end - start + 1)
        end += 1
    return max_len
```

Time complexity?

O(N)

Space complexity?

Time / Space complexity

```
def longest_substring_two_distinct(s):
    char_counts = defaultdict(int)
    start, end, max_len = 0, 0, 0

    while end < len(s):
        # Fetch the newest character in the substring and update its count
        char_counts[s[end]] += 1

        # Too many distinct characters in the substring, we need to shrink the window
        while len(char_counts) > 2:
            start_char = s[start]
            char_counts[start_char] -= 1
            start += 1
            if char_counts[start_char] == 0:
                del char_counts[start_char]
        max_len = max(max_len, end - start + 1)
        end += 1
    return max_len
```

Time complexity?

O(N)

Space complexity?

O(1)

Solutions

Python and Java solutions are posted!

Recommendation: Try this problem on your own before looking at the solution to practice

In class exercises

In class exercises

Shifting Letters

Set matrix zeros

Longest substring without repeating characters

Group anagrams

Recap

Survey

Please take the short survey for this week!

Wrap up

Shifting Letters, converting string to ascii

Set matrix zeros, matrix traversal

Longest substring without repeating characters, sliding window

Group anagrams, string