KAMK-AI-Studies / Tietorakenteet ja Algoritmit TTM23SAI

Essential Algorithms : A Practical Approach to Computer Algorithms

Tehtävät koodataan jupyterlab tai notebook alustalla ja palautetaan yhtenä etunimi_sukunimi_tehtävät1.ipynb tiedostona erillään muista tehtävä palautuksista Tehtävä palautukset laatikkoon.

Koodin tulee olla toimiva, siistitty ja kommentoitu. Näillä tehtävillä saavutat arvosanan 1.

Tehtävien pseudokoodit löydät kurssi kirjasta alla olevista sivunumeroista, tehtäväsi on tehdä näistä omaa toimivaa python koodia ja käyttää mahdollisimman vähän valmiita funktioita. s.09 Contains Duplicates s.20 Exercises 1 Contains Duplicates Improved Version s.31 Randomized array s.55-63 Linked List s.63-66 Double Linked List s.68 Insertionsort s.87 FindMinimum s.87 FindMaximum s.87 FindAverage s.87 FindMedian s.112 Push Stack s.112 Pop Stack s.117 Reverse Array with a Stack

ContainsDublicates

In [1]:
```
%time
def containsDublicates(lst):
    for i in range(len(lst)):
        for j in range(len(lst)):
            if i != j and lst[i] == lst[j]:
                return True
    return False

#containsDublicates([1,2,3,4,5,6])
containsDublicates([1,2,3,4,5,5])
```

```
CPU times: user 1 µs, sys: 1 µs, total: 2 µs
Wall time: 2.62 µs
```

Out[1]: True

In [2]:
```
#Optimoitu vastaus
```

In [3]:
```
%time
def contains_duplicates(lst):
    return len(lst) != len(set(lst))
```

```
#containsDublicates([1,2,3,4,5,6])
containsDublicates([1,2,3,4,5,5])
```

```
CPU times: user 1e+03 ns, sys: 0 ns, total: 1e+03 ns
Wall time: 2.86 µs
```

Out[3]: True

Duplicates Improved Version

In [4]:
```
%time
def contains_duplicates(lst):
    for i in range(len(lst) - 1):
        for j in range(i + 1, len(lst)):
            if lst[i] == lst[j]:
                return True
    return False


containsDublicates([1,2,3,4,5,5])
#print(contains_duplicates([1, 2, 3, 2]))
```

```
CPU times: user 0 ns, sys: 1 µs, total: 1 µs
Wall time: 3.58 µs
```

Out[4]: True

Pick Item with Probabilities (extra)

In [5]:
```
import random

def pick_item_with_probabilities(items, probabilities):
    value = random.random()
    for i in range(len(items)):
        value -= probabilities[i]
        # If the value drops to zero or below, select the current item
        if value <= 0:
            return items[i]

items = ["red", "green", "blue"]
probabilities = [0.25, 0.3, 0.45]

result = pick_item_with_probabilities(items, probabilities)
print(f"Selected item: {result}")
```

```
Selected item: red
```

Randomized Array

In [6]:
```
import random

def randomize_array(array):
```

```
    max_i = len(array)
    for i in range(max_i):
        j = random.randint(i, max_i - 1)
        array[i], array[j] = array[j], array[i]

my_array = ["a", "b", "c", "d", "e"]
randomize_array(my_array)
print(my_array)
```

```
['e', 'a', 'b', 'd', 'c']
```

In [7]:
```
%time
import math
result = math.pow(7, 6)
print(result)  # Tulostaa: 343.0 (palauttaa aina liukuluvun)
```

```
CPU times: user 2 µs, sys: 1 µs, total: 3 µs
Wall time: 4.53 µs
117649.0
```

In [8]:
```
%time
import math
result = math.pow(7, 2) * math.pow(7, 2) * math.pow(7, 2)
print(result)
```

```
CPU times: user 1 µs, sys: 0 ns, total: 1 µs
Wall time: 3.81 µs
117649.0
```

finding zeros (extra)

In [9]:
```
def find_zero(f, dfdx, initial_guess, max_error, max_iterations=100):
    x = initial_guess
    for i in range(max_iterations):
        y = f(x)
        if abs(y) < max_error:
            break
        x = x - y / dfdx(x)
    return x
```

In [10]:
```
# Määritellään funktio ja sen derivaatta
def f(x):
    return x**2 - 2

def dfdx(x):
    return 2 * x

# Lähtöarvaus ja toleranssi
initial_guess = 1.0
max_error = 1e-6

# Lasketaan juuri
root = find_zero(f, dfdx, initial_guess, max_error)
print(f"Juuri: {root}")
```

Juuri: 1.4142135623746899

Linked List

In [11]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def iterate(top):
    while top is not None:  # Loop through the linked list
        print(top.value)    # Print the value of the current node
        top = top.next      # Move to the next node


# Creating nodes
node1 = Node(10)
node2 = Node(20)
node3 = Node(30)

# Linking nodes
node1.next = node2  # Node 1 points to Node 2
node2.next = node3  # Node 2 points to Node 3

# Node 3 is the last node, so its "next" is already None

iterate(node1)
```

```
10
20
30
```

Finding Cells

In [12]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def FindCell(top, target):
    position = 1
    while top is not None:  # Traverse the linked list
        if top.value == target:  # Check if the current node's value matches the ta
            return top, position  # Return the current node if a match is found
        top = top.next  # Move to the next node
        position += 1 #Increment the position counter
    # If the loop ends without finding the target, return None
    return None, -1

# Creating nodes
node1 = Node(10)
node2 = Node(20)
```

```
node3 = Node(30)

# Linking nodes to form a linked list
node1.next = node2  # Node 1 points to Node 2
node2.next = node3  # Node 2 points to Node 3

# Node 3 is the last node, so its "next" is already None

# Searching for a value in the linked list
target_value = 20
result, position = FindCell(node1, target_value)

if result is not None:
    print(f"Node with value {target_value} found: Node {position}")
else:
    print(f"Node with value {target_value} not found.")
```

```
Node with value 20 found: Node 2
```

In [13]:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def FindCellBefore(top, target):
    # If the list is empty, or the target is in the first node, return None
    if top is None or top.next is None:
        return None

    # Traverse the list looking for the node before the target node
    while top.next is not None:
        if top.next.value == target:  # Check if the next node contains the target
            return top  # Return the current node (before the target node)
        top = top.next  # Move to the next node

    # If no match is found, return None
    return None

# Creating nodes
node1 = Node(10)
node2 = Node(20)
node3 = Node(30)

# Linking nodes to form a linked list
node1.next = node2  # Node 1 points to Node 2
node2.next = node3  # Node 2 points to Node 3

# Node 3 is the last node, so its "next" is already None

# Searching for the node before a target value
target_value = 30
result = FindCellBefore(node2, target_value)

if result is not None:
    print(f"Node before the one with value {target_value} has value {result.value}.
```

```
else:
    print(f"No node found before the one with value {target_value}.")
```

```
Node before the one with value 30 has value 20.
```

In [14]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def FindCellBefore(top, target):
    # Traverse the list looking for the node before the target node
    while top is not None and top.next is not None:  # Ensure we don't access None
        if top.next.value == target:  # Check if the next node contains the target
            return top  # Return the current node (the one before the target node)
        top = top.next  # Move to the next node

    # If no match is found, return None
    return None

# Creating nodes
node1 = Node(10)
node2 = Node(20)
node3 = Node(30)

# Linking nodes to form a linked list
node1.next = node2  # Node 1 points to Node 2
node2.next = node3  # Node 2 points to Node 3

# Node 3 is the last node, so its "next" is already None

# Searching for the node before a target value
target_value = 30
result = FindCellBefore(node1, target_value)

if result is not None:
    print(f"Node before the one with value {target_value} has value {result.value}.
else:
    print(f"No node found before the one with value {target_value}.")
```

```
Node before the one with value 30 has value 20.
```

Using Sentinels

Adding Cells at the Beginning

In [15]:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def FindCellBeforeWithSentinel(top, target):
    # Create a sentinel node that points to the head of the list
```

```
    sentinel = Node(None)  # Dummy value
    sentinel.next = top

    # Use the sentinel as the starting point
    current = sentinel

    while current.next is not None:  # Traverse the list
        if current.next.value == target:  # Check if the next node contains the tar
            return current  # Return the node before the target
        current = current.next  # Move to the next node

    # If no match is found, return None
    return None

# Creating nodes
node1 = Node(10)
node2 = Node(20)
node3 = Node(30)

# Linking nodes to form a linked list
node1.next = node2  # Node 1 points to Node 2
node2.next = node3  # Node 2 points to Node 3

# Node 3 is the last node, so its "next" is already None

# Searching for the node before a target value
target_value = 10
result = FindCellBeforeWithSentinel(node1, target_value)

if result is not None:
    print(f"Node before the one with value {target_value} has value {result.value}.
else:
    print(f"No node found before the one with value {target_value}.")
```

```
Node before the one with value 10 has value None.
```

Adding Cells at the End

In [16]:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def AddAtEnd(sentinel, new_cell):
    # Traverse the list to find the last cell
    while sentinel.next is not None:
        sentinel = sentinel.next  # Move to the next node

    # Add the new cell at the end
    sentinel.next = new_cell
    new_cell.next = None  # Ensure the new cell's `next` is set to None

# Creating nodes for testing
node1 = Node(10)
```

```
node2 = Node(20)
node3 = Node(30)

# Linking nodes to form a linked list
node1.next = node2  # Node 1 points to Node 2
node2.next = node3  # Node 2 points to Node 3

# Creating a new cell to add at the end
new_node = Node(40)

# Adding the new cell at the end of the list
AddAtEnd(node1, new_node)

# Verifying the result by traversing the list
current = node1
while current is not None:
    print(current.value)
    current = current.next
```

```
10
20
30
40
```

Inserting Cells After Other Cells

In [17]:
```
def InsertCell(after_me, new_cell):
    new.cell.next = after_me.next
    after_me.next = new_cell
```

In [18]:
```
def DeleteAfter(after_me):
    target = after_me.next
    after_me = target.next
```

Delete list

In [19]:
```
def DestroyList(sentinel):
    while sentinel is not None:  # Traverse the list
        # Save a reference to the next cell
        next_cell = sentinel.next

        # "Destroy" the current node
        sentinel.next = None  # Sever the link (optional, but good practice)
        sentinel = None       # Clear the current node (optional for clarity)

        # Move to the next cell
        sentinel = next_cell
```

Double Linked List

In [20]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
        self.prev = None

def InsertCell(after_me, new_cell):
    # Update `next` links
    new_cell.next = after_me.next  # Link the new node to the next node
    after_me.next = new_cell       # Link the current node to the new node

    # Update `prev` links
    if new_cell.next is not None:  # If there is a next node, update its `prev` poi
        new_cell.next.prev = new_cell
    new_cell.prev = after_me  # Link the new node back to the current node
```

Sorted linked list

In [21]:
```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

def InsertCell(sentinel, new_cell):
    # Find the appropriate position to insert the new cell
    while sentinel.next is not None and sentinel.next.value < new_cell.value:
        sentinel = sentinel.next  # Move to the next node

    # Insert the new cell after sentinel
    new_cell.next = sentinel.next
    sentinel.next = new_cell
```

In [22]:
```python
def InsertCell(sentinel, new_cell):
    while sentinel.next is not None and sentinel.next.value < new_cell.value:
        sentinel = sentinel.next  # Move to the next node
    new_cell.next = sentinel.next
    sentinel.next = new_cell
```

Insertionsort

In [23]:
```python
class Node:
    def __init__(self, value=None):
        self.value = value
        self.next = None

def Insertionsort(old_sentinel):
    # Make a sentinel for the sorted list
    new_sentinel = Node()  # New sentinel node
    new_sentinel.next = None

    # Skip the old list's sentinel
```

```
    old_sentinel = old_sentinel.next

    # Repeat until all items are added to the new list
    while old_sentinel is not None:
        # Get the next cell to add to the sorted list
        next_cell = old_sentinel

        # Advance old_sentinel for the next iteration
        old_sentinel = old_sentinel.next

        # Find where to insert the cell in the sorted list
        after_me = new_sentinel
        while after_me.next is not None and after_me.next.value < next_cell.value:
            after_me = after_me.next

        # Insert the cell into the sorted list
        next_cell.next = after_me.next
        after_me.next = next_cell

    # Return the sorted list
    return new_sentinel
```

FindMinimum

In [24]:
```
def FindMinimum(array):
    # Initialize the minimum with the first element of the array
    minimum = array[0]

    # Loop through the rest of the array
    for i in range(1, len(array)):
        if array[i] < minimum:
            minimum = array[i]  # Update minimum if a smaller value is found

    return minimum

array = [34, 78, 12, 56, 89, 3, 45]
minimum_value = FindMinimum(array)
print("The minimum value in the array is:", minimum_value)
```

```
The minimum value in the array is: 3
```

In [25]:
```
def FindMinimum(array):
    return min(array)

array = [34, 78, 12, 56, 89, 3, 45]
print("The minimum value in the array is:", FindMinimum(array))
```

```
The minimum value in the array is: 3
```

FindMaximum

In [26]:
```python
def FindMaximum(array):
    # Initialize the maximum with the first element of the array
    maximum = array[0]

    # loop through the rest of array
    for i in range (i, len(array)):
        if array[i] > maximum:
            maximum = array[i]  # Update maximum if a smaller value is found
    return maximum

array = [34, 78, 12, 56, 89, 3, 45]
print("The minimum value in the array is:", FindMinimum(array))
```

```
The minimum value in the array is: 3
```

In [27]:
```python
def FindMinimum(array):
    return max(array)

array = [34, 78, 12, 56, 89, 3, 45]
print("The minimum value in the array is:", FindMinimum(array))
```

```
The minimum value in the array is: 89
```

FindAverage

In [28]:
```python
def FindAverage(array):
    # Initialize total to 0
    total = 0

    # Loop through the array and sum the elements
    for i in range(len(array)):
        total += array[i]

    # Return the average
    return total / len(array)


array = [34, 78, 12, 56, 89, 3, 45]
print(f"The average value of the array is:{FindAverage(array):.2f}")
```

```
The average value of the array is:45.29
```

In [29]:
```python
def FindMinimum(array):
    return avg(array)

array = [34, 78, 12, 56, 89, 3, 45]
print(f"The average value of the array is:{FindAverage(array):.2f}")
```

```
The average value of the array is:45.29
```

FindMedian

In [30]:
```python
def FindMedian(array):
    for i in range(len(array)):
        # Count numbers smaller and larger than array[i]
        num_larger = 0
        num_smaller = 0

        for j in range(len(array)):
            if array[j] < array[i]:
                num_smaller += 1
            if array[j] > array[i]:
                num_larger += 1

    # Check if current element is the median
    if num_smaller == num_larger:
        return array[i]

array = [34, 78, 12, 56, 89, 3, 45]
median_value = FindMedian(array)
print("The median value of the array is:", median_value)
```

```
The median value of the array is: 45
```

In [31]:
```python
def FindMedian(array):
    # Sort the array
    array.sort()

    # Check if the length of the array is odd
    if len(array) % 2 == 1:
        # Return the middle element
        return array[len(array) // 2]
    else:
        # Return the average of the two middle elements
        mid = len(array) // 2
        return (array[mid - 1] + array[mid]) / 2

array = [34, 78, 12, 56, 89, 3, 45]
median_value = FindMedian(array)
print("The median value of the array is:", median_value)
```

```
The median value of the array is: 45
```

Push Stack

In [32]:
```python
class Node:
    def __init__(self, value=None):
        self.value = value
        self.next = None

def Push(sentinel, new_value):
    # Create a new node to hold the value
    new_cell = Node(new_value)

    # add the new cell to the front of the linked list
```

```
    new_cell.next = sentinel.next
    sentinel.next = new_cell
```

In [33]:
```
# Helper function to print the linked list
def print_linked_list(sentinel):
    current = sentinel.next  # Skip the sentinel
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Initialize a sentinel node (dummy head)
sentinel = Node()

# Push values into the linked list
Push(sentinel, 10)
Push(sentinel, 20)
Push(sentinel, 30)

# Print the linked list
print_linked_list(sentinel)
```

```
30 -> 20 -> 10 -> None
```

Pop Stack

In [34]:
```
class Node:
    def __init__(self, value=None):
        self.value = value
        self.next = None

def Pop(sentinel):
    # Ensure there is an item to pop
    if sentinel.next is None:
        raise Exception("Cannot pop from an empty list.")

    # Get the top cell's value
    result = sentinel.next.value

    # Remove the top cell from the linked list
    sentinel.next = sentinel.next.next

    # Return the result
    return result
```

In [35]:
```
# Helper function to print the linked list
def print_linked_list(sentinel):
    current = sentinel.next  # Skip the sentinel
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

# Initialize a sentinel node (dummy head)
```

```
sentinel = Node()

# Push values into the linked list
Push(sentinel, 10)
Push(sentinel, 20)
Push(sentinel, 30)

print("Initial linked list:")
print_linked_list(sentinel)

# Pop values from the list
print("\nPopped value:", Pop(sentinel))
print("Linked list after pop:")
print_linked_list(sentinel)

print("\nPopped value:", Pop(sentinel))
print("Linked list after pop:")
print_linked_list(sentinel)
```

```
Initial linked list:
30 -> 20 -> 10 -> None

Popped value: 30
Linked list after pop:
20 -> 10 -> None

Popped value: 20
Linked list after pop:
10 -> None
```

Reverse Array with a Stack

In [36]:
```
def ReverseArray(values):
    # Use a list to simulate a stack
    stack = []

    # Push the values from the array onto the stack
    for value in values:
        stack.append(value)

    # Pop the items off the stack into the array
    for i in range(len(values)):
        values[i] = stack.pop()  # Pop from the stack and overwrite the array
```

In [37]:
```
values = [1, 2, 3, 4, 5]

print("Original array:", values)

ReverseArray(values)

print("Reversed array:", values)
```

```
Original array: [1, 2, 3, 4, 5]
Reversed array: [5, 4, 3, 2, 1]
```

In [38]:
```
def ReverseArray(values):
    values.reverse()  # In-place reversal

values = [1, 2, 3, 4, 5]

print("Original array:", values)
ReverseArray(values)
print("Reversed array:", values)
```

```
Original array: [1, 2, 3, 4, 5]
Reversed array: [5, 4, 3, 2, 1]
```

In [39]:
```
def ReverseArray(values):
    values[:] = values[::-1]  # Slice-based reversal

values = [1, 2, 3, 4, 5]

print("Original array:", values)
ReverseArray(values)
print("Reversed array:", values)
```

```
Original array: [1, 2, 3, 4, 5]
Reversed array: [5, 4, 3, 2, 1]
```