Data Structure Assignment 2

إعداد: رنا خميس

Theoretical Questions (PDF)

Question: What is the differences between Adjacency List and Adjacency Matrix representation of Graph?

Feature	Adjacency Matrix	Adjacency List
Representation	A 2D array of size V x V where V is the number of vertices.	An array of lists where each index represents a vertex and its list contains neighbors.
Space Complexity	$O(V^2)$ - High, especially for sparse graphs.	$O(V + E)$ - Efficient for sparse graphs.
Edge Search	$O(1)$ - Fast to check if an edge exists between two vertices.	$O(V)$ - Slower, as you must traverse the list.
Adding Vertex	$O(V^2)$ - Requires resizing the entire matrix.	$O(1)$ - Simply add a new list.
Sparse Graphs	Wasteful of space.	Very efficient.

Lab Questions (CODE)

1. Reverse a string using Stack

```python
def reverse_string(input_str):
    stack = []
    for char in input_str:
        stack.append(char)

    reversed_str = ""
    while stack:
        reversed_str += stack.pop()
    return reversed_str

# Example Usage
print(reverse_string("Hello")) # Output: olleH
```

2. Sort a stack using only another Stack

```python
def sort_stack(stack):
    temp_stack = []
    while stack:
        current = stack.pop()
        while temp_stack and temp_stack[-1] > current:
            stack.append(temp_stack.pop())
        temp_stack.append(current)
    return temp_stack

# Example Usage
my_stack = [34, 3, 31, 98, 92, 23]
print(sort_stack(my_stack)) # Output: [3, 23, 31, 34, 92, 98]
```

3. Reverse the order of elements in a queue

```python
from collections import deque

def reverse_queue(queue):
    stack = []
    while queue:
        stack.append(queue.popleft())
    while stack:
        queue.append(stack.pop())
    return queue

# Example Usage
q = deque([1, 2, 3, 4, 5])
print(reverse_queue(q)) # Output: deque([5, 4, 3, 2, 1])
```

4. Implement a priority queue where the smallest element is dequeue first

```
import heapq

class PriorityQueue:
    def __init__(self):
        self.elements = []

    def enqueue(self, item):
        heapq.heappush(self.elements, item)

    def dequeue(self):
        if not self.is_empty():
            return heapq.heappop(self.elements)
        return None

    def is_empty(self):
        return len(self.elements) == 0

# Example Usage
pq = PriorityQueue()
pq.enqueue(10)
pq.enqueue(5)
pq.enqueue(20)
print(pq.dequeue()) # Output: 5 (Smallest)
```

5. Merge two sorted queues into a single sorted queue

```python
from collections import deque

def merge_sorted_queues(q1, q2):
    merged_queue = deque()
    while q1 and q2:
        if q1[0] < q2[0]:
            merged_queue.append(q1.popleft())
        else:
            merged_queue.append(q2.popleft())

    while q1:
        merged_queue.append(q1.popleft())
    while q2:
        merged_queue.append(q2.popleft())

    return merged_queue

# Example Usage
queue1 = deque([1, 3, 5])
queue2 = deque([2, 4, 6])
print(merge_sorted_queues(queue1, queue2)) # Output: deque([1, 2, 3, 4, 5,
6])
```

Submission Notes

- **GitHub Repository:** Assignments
- **Package Name:** assignment_1
- **Author:** رنا خميس
- **Status:** Complete and Ready for Submission