IARE

IARE
INSTITUTE OF AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : RAGHERLA SANTHOSH

Roll Number									
2	3	9	5	1	A	1	2	G	3

Class : IT-B Semester : 03

Course Code : AGSD10 Course Name : OS Laboratory

Name of the Course Faculty : Mr. N. Raghava Rao Faculty ID : IARE10924

Exercise Number : 10 Week Number : 10 Date : 22/11/2024

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure	Performance in the Lab	Source Code	Calculations and Graphs	Program Execution	Results and Error Analysis	Viva - Voce	Total
			4	4		4		4		4	20
1	10.1	The Story of Busy Cafe and its Orders	4		4	4		4		3	19
2	10.2	The Tale of the Library and its Book Shelves.									
3	10.3	The Story of Busy Cafe and its Special Recipe									
4	10.4	The Tale of the Art Gallery & its exhibition									
5	10.5	The Tale of the Library and its popular books.									
6											
7											
8											
9											
10											
11											
12											

Santhosh.

Signature of the Student

Signature of the Faculty

10. Page Replacement Algorithms.

10.1 The Story Of a Busy Cafe & its Orders.

AIM :- Write a program for Cafe FIFO to manage their incoming orders, ensuring that the first order received is also the first one to be processed.

PROGRAM :-

```
from collections import deque
class CafeFIFO:
    def __init__(self, max_capacity=4):
        self.counter = deque(maxlen=max_capacity)
        self.max_capacity = max_capacity
    def add_order(self, order):
        """Add a new order to the counter (FIFO queue)"""
        if len(self.counter) == self.max_capacity:
            removed_order = self.counter.popleft()
            Print(f"Counter full. Processing & removing oldest
                order: {removed_order}")
```

```
        self.counter.append(order)
        print(f"Added order: {order}")
    def display_orders(self):
        """Display current orders on the counter"""
        if self.counter:
            print("Current orders on the counter:", ", ".join
                (self.counter))
        else:
            print("No orders on the counter")
    def Process_remaining_orders(self):
        """Process remaining orders in the queue (FIFO)"""
        while self.counter:
            order = self.counter.popleft()
            print(f"Processing order: {order}")
    def simulate_cafe():
        cafe = CafeFIFO(max_capacity = 4)
        orders = ['Coffee', "Tea", "Muffin", "Croissant",
                  "Smoothie"]
```

```
for order in orders :
    cafe.add_order(order)
    cafe.display_orders()
Print("\nEnd of the day - Processing remaining orders:")
cafe.process_remaining_orders()
Print("\nFinal orders on the counter (Should be empty):")
cafe.display_orders()
if __name__ == "__main__":
    Simulate_cafe()
```

OUTPUT :-

The program is Executed Successfully.

10.2 The tale of the Library and its Book Shelves.

AIM :- Write a Program for the library of Timeless Tales uses the FIFO Policy to manage the books on the shelf.

PROGRAM :-

```
from collections import deque
class CafeFIFO :
    def __init__(self, max_capacity = 4) :
        self.counter = deque(maxlen = max_capacity)
        self.max_capacity = max_capacity
    def add_order(self, order) :
        """ Add a new order to the counter (FIFO queue)"""
        if len(self.counter) == self.max_capacity :
            removed_order = self.counter.popleft()
            Print(f" Counter full. Procesing & removing oldest
                order : {removed_order}")
        self.counter.append(order)
        Print(f" Added order : {order}")
    def display_orders(self) :
        """ Display current orders on the counter."""
        if self.counter :
            Print(" Current Orders on the counter:", ", ".
```

```
        join (self.counter))
    else:
        Print ("No orders on the counter")
def Process_remaining_orders(self):
    """ Process remaining orders in the queue (FIFO) """
    while self.counter:
        order = self.counter.popleft()
        Print (f"Procesing order: {order}")
def Simulate_cafe():
    cafe = CafeFIFO (max_capacity = 4)
    orders = ["Coffee", "Tea", "Muffin", "Croissant",
              "Smoothie"]
    for order in orders:
        cafe.add_order (order)
        cafe.display_orders()
Print ("\nEnd of the day - Processing remaining order:")
cafe.process_remaining_orders()
```

```
Print ("In Final orders on the counter (should be empty):")
cafe.display_orders()
if __name__ == "__main__":
    simulate_cafe()
```

OUTPUT :-

The program is Executed Successfully.

10.3 The Story Of the Busy Cafe & Its Special Recipe Book :

AIM :- Write a Program for Cafe Nostalgia, It uses the LRU Policy to manage her recipe book on the counter.

PROGRAM :-

```
from collections import OrderedDict
class RecipeBook:
    def __init__(self, max_capacity = 4):
        self.counter = OrderedDict()
```

```
def access_recipe(self, recipe):
    """Simulate accessing a recipe."""
    if recipe in self.counter:
        self.counter.move_to_end(recipe)
        Print(f"Recipe '{recipe}' is already on the counter.
              Moved to the most recent")
    else:
        if len(self.counter) == self.max_capacity:
            removed_recipe = self.counter.popitem(last=False)
            Print(f"Counter full. Removed least recently used
                  recipe: '{removed_recipe[0]}'")
        self.counter[recipe] = None
        Print(f"Added recipe: '{recipe}'")
def display_counter(self):
    """Display the current recipes on the counter
       in the order they are accessed."""
    Print("Current recipes on the counter:",
          list(self.counter.keys()))
```

```
def simulate_cafe():
    recipe_book = RecipeBook(max_capacity = 4)
    recipes = [
        "Pumpkin Spice Latte",
        "Apple pie",
        "Blueberry Muffin",
        "Cinnamon Roll",
        "Apple Pie",
        "Maple Pancakes",
        "Pumpkin Spice Latte"
    ]
    for recipe in recipes:
        recipe_book.access_recipe(recipe)
        recipe_book.display_counter()
    Print("In Final recipes on the counter")
    recipe_book.display_counter()
if __name__ == "__main__":
```

Simulate_cafe().

OUTPUT :-

The Program is Executed successfully.

10.4 The Tale Of the Art Gallery and Its Exhibition.

AIM :- Write a Program for Gallery Of Modern Arts to manage the display using the LRU policy.

PROGRAM :-

```
from collections import deque
class GalleryDisplay:
    def __init__(self, capacity):
        self.capacity = capacity
        self.display_board = deque()
    def access_artwork(self, artwork):
        if artwork in self.display_board:
            self.display_board.remove(artwork)
```

```
elif len(self.display_board) == self.capacity:
    self.display_board.popleft()
self.display_board.append(artwork)
def current_display(self):
    return list(self.display_board)
gallery = GalleryDisplay(capacity=6)
artwork_requests = [
"Sunset Over the Lake",
"Abstract Dreams",
"Cityscape at Dusk",
"Golden Fields",
"Harmony in Blue",
]
for artwork in artwork_requests:
    gallery.access_artwork(artwork)
Print("Final State of the display board:")
```

```
for artwork in gallery.current_display():
    Print(artwork)
```

OUTPUT :-

The program is Executed Successfully.

10.5 **The Tale Of the library and Its Popular Books .**

AIM :- Write a Program for the library manager of the Book Haven to manage the Display using LFU policy.

PROGRAM :-

```
from collections import defaultdict, OrderedDict
class LFUCache:
    def __init__(self, capacity):
        self.capacity = capacity
        self.freq_map = defaultdict(OrderedDict)
        self.book_freq = {}
        self.min_freq = 0
```

```
def access_book(self, book):
    if book in self.book_freq:
        self._update_frequency(book)
    else:
        if len(self.book_freq) == self.capacity:
            self._evict_book()
        self.freq_map[1][book] = None
        self.book_freq[book] = 1
        self.min_freq = 1

def _update_frequency(self, book):
    freq = self.book_freq[book]
    del self.freq_map[freq][book]
    if not self.freq_map[freq]:
        del self.freq_map[freq]
        if freq == self.min_freq:
            self.min_freq += 1
    self.freq_map[freq+1][book] = None
```

```
        self.book_freq[book] += 1

    def _evict_book(self):
        book, _ = self.freq_map
        if not self.freq_map[self.min_freq]:
            del self.freq_map[self.min_freq]
        del self.book_freq[book]

    def current_display(self):
        display = []
        for freq in sorted(self.freq_map.keys(),
                           reverse = True):
            display.extend(list(self.freq_map[freq].
                                keys()))
        return display

library_display = LFUCache(capacity=5)
book_checkouts = [
    ("The Great Gatsby", 3),
    ("1984", 5),
```

```
("To Kill a Mockingbird", 2),
("Pride and prejudice", 4),
("The Catcher in the Rye", 1),
("Moby Dick", 6),
("The Odyssey", 3),
("War and Peace", 2).
]

for book, frequency in book_checkouts:
for _ in range (frequency):
    library_display.access_book(book)
Print("Final State of the Display board:")
for book in library_display.current_
    display():
    Print(book).
```

OUTPUT :-

The Program Executed Successfully.