Bahria University,

Karachi Campus



LAB EXPERIMENT NO.

\_\_\_06\_\_\_

LIST OF TASKS

|  |  |
| --- | --- |
| TASK NO | OBJECTIVE |
| 1 | Create a program that recursively searches for files with a specific extension in a given directory. |
| 2 | Implement Fibonacci series using recursive function. |
| 3 | Implement Undo Mechanism Using Appropriate ADT |
| 4 | Identify ADT used in Print Job Management and implement its code as well. |
| 5 | You are working on a social network analysis project and need to implement various set operations to analyze relationships between users. Your task is to implement a program that performs the following functionalities using the Set ADT:Friends of Friends: Given a user, find their friends of friends (i.e., users who are friends with at least one of their friends but not directly connected to the user).Mutual Friends: Given two users, find their mutual friends (i.e., users who are friends with both users).Friend Suggestions: Given a user, find friend suggestions by identifying users who are friends with at least two of the user's friends but are not yet connected to the user. |
| 6 | Implement a gradebook analysis program using the Map ADT. The program should read a CSV file containing student names and their corresponding grades. The CSV file will have two columns: "Student Name" and "Grade". The program should analyze the grades and provide the following functionalities:Calculate the average grade for all students.Determine the student with the highest grade.Determine the student with the lowest grade.Count the number of students who received a grade above a specified threshold. |

Submitted On

8-11-2023

(Date: DD/MM/YY)

**Task 1:** Create a program that recursively searches for files with a specific extension in a given directory.

**Solution:**

import os

def find\_files(dir\_path, extension):

for file\_or\_dir in os.listdir(dir\_path):

full\_path = os.path.join(dir\_path, file\_or\_dir)

if os.path.isdir(full\_path):

find\_files(full\_path, extension)

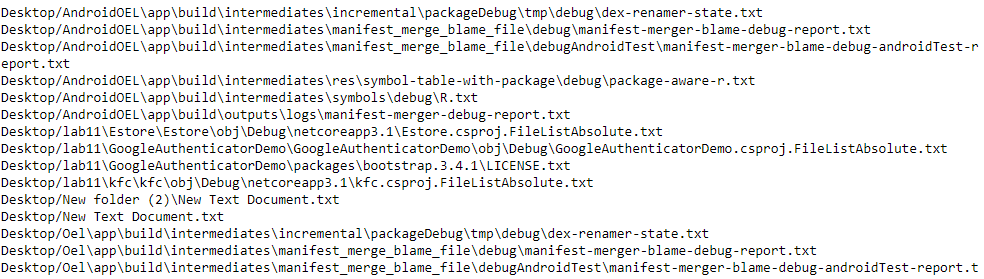
else:

if full\_path.endswith(extension):

print(full\_path)

find\_files('Desktop/', '.txt')

**Output:**



**Task 2:** Implement Fibonacci series using recursive function.

**Solution:**

def recur\_fibo(n):

if n <= 1:

return nelse:

return(recur\_fibo(n-1) + recur\_fibo(n-2))

nterms = int(input("How many numbers do you want to write? "))

if nterms <= 0:

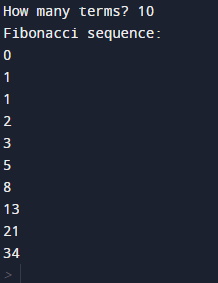
print("Plese enter a positive integer")else:

print("Fibonacci sequence:")

for i in range(nterms):

print(recur\_fibo(i))

**Output:**



**Task 3:** Implement Undo Mechanism Using Appropriate ADT

**Solution:**

class UndoStack:

def \_\_init\_\_(self):

self.stack = []

def push(self, state):

self.stack.append(state)

def pop(self):

if not self.is\_empty():

return self.stack.pop()

else:

return None

def is\_empty(self):

return len(self.stack) == 0

undo\_stack = UndoStack()

document\_state = "Initial Content"

while True:

action = input("Enter an action (Type 'undo' to undo or 'exit' to exit): ")

if action == "exit":

break

elif action == "undo":

previous\_state = undo\_stack.pop()

if previous\_state is not None:

document\_state = previous\_state

print(f"Undo: Document state is now '{document\_state}'")

else:

print("Nothing to undo.")

else:

undo\_stack.push(document\_state)

document\_state = action

print(f"Document state is now '{document\_state}'")

A screenshot of a computer code

Description automatically generated**Output:**

shoaib

shoaib

shoaib

shoaib

shoaib

shoaib

shoaib

shoaib

**Task 4:** Identify ADT used in Print Job Management and implement its code as well.

**Solution:**

import queue

def print\_job\_management():

print\_queue = queue.Queue()

while True:

print("Options:")

print("1. Submit a print job")

print("2. Print a job")

print("3. Exit")

choice = input("Enter your choice: ")

if choice == "1":

job = input("Enter the print job: ")

print\_queue.put(job)

print(f"Print job '{job}' has been added to the queue.")

elif choice == "2":

if not print\_queue.empty():

job = print\_queue.get()

print(f"Printing: {job}")else:

print("No print jobs in the queue.")

elif choice == "3":

breakelse:

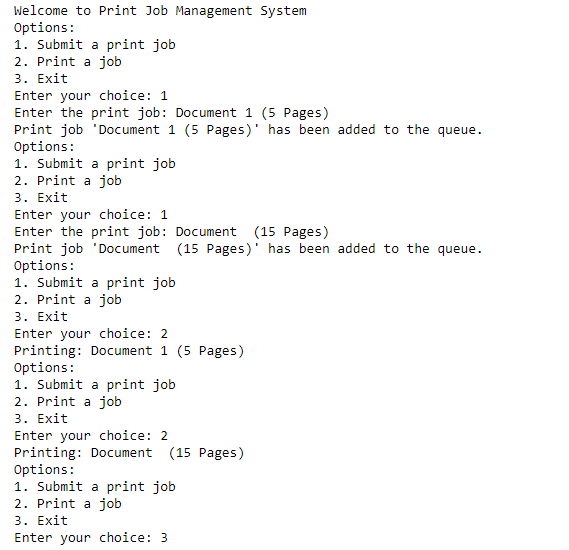
print("Invalid choice. Please choose 1, 2, or 3.")

if \_\_name\_\_ == "\_\_main\_\_":

print("Welcome to Print Job Management System")

print\_job\_management()

**Output:**



* **Task 5:** You are working on a social network analysis project and need to implement various set operations to analyze relationships between users. Your task is to implement a program that performs the following functionalities using the Set ADT:

1. Friends of Friends: Given a user, find their friends of friends (i.e., users who are friends with at least one of their friends but not directly connected to the user).
2. Mutual Friends: Given two users, find their mutual friends (i.e., users who are friends with both users).
3. Friend Suggestions: Given a user, find friend suggestions by identifying users who are friends with at least two of the user's friends but are not yet connected to the user.

**Solution:**

class SocialNetwork:

def \_\_init\_\_(self):

# Create a dictionary to store the relationships between users.

# The keys are user IDs, and the values are sets of their friends.

self.users = {}

def add\_user(self, user\_id):

# Initialize a new user with an empty set of friends.

self.users[user\_id] = set()

def add\_friend(self, user\_id, friend\_id):

# Add a friend to the user's friend set.

if user\_id in self.users and friend\_id in self.users:

self.users[user\_id].add(friend\_id)

self.users[friend\_id].add(user\_id)

def friends\_of\_friends(self, user\_id):

if user\_id not in self.users:

return set() # User doesn't exist, return an empty set.

user\_friends = self.users[user\_id]

friends\_of\_friends\_set = set()

for friend in user\_friends:

friends\_of\_friends\_set.update(self.users[friend])

# Remove the user and their direct friends from the set.

friends\_of\_friends\_set.discard(user\_id)

friends\_of\_friends\_set.difference\_update(user\_friends)

return friends\_of\_friends\_set

def mutual\_friends(self, user1, user2):

if user1 not in self.users or user2 not in self.users:

return set() # User(s) don't exist, return an empty set.

mutual\_friends\_set = self.users[user1].intersection(self.users[user2])

return mutual\_friends\_set

def friend\_suggestions(self, user\_id):

if user\_id not in self.users:

return set() # User doesn't exist, return an empty set.

user\_friends = self.users[user\_id]

friend\_suggestions\_set = set()

for friend in user\_friends:

for suggested\_friend in self.users[friend]:

if suggested\_friend != user\_id and suggested\_friend not in user\_friends:

friend\_suggestions\_set.add(suggested\_friend)

return friend\_suggestions\_set

if \_\_name\_\_ == '\_\_main\_\_':

social\_network = SocialNetwork()

social\_network.add\_user(1)

social\_network.add\_user(2)

social\_network.add\_user(3)

social\_network.add\_user(4)

social\_network.add\_friend(1, 2)

social\_network.add\_friend(2, 3)

social\_network.add\_friend(3, 4)

social\_network.add\_friend(4, 1)

print("Friends of Friends for User 1:", social\_network.friends\_of\_friends(1))

print("Mutual Friends between User 1 and User 3:", social\_network.mutual\_friends(1, 3))

print("Friend Suggestions for User 1:", social\_network.friend\_suggestions(1))

**Output:**

A group of black text

Description automatically generated

**Task 6:** Implement a gradebook analysis program using the Map ADT. The program should read a CSV file containing student names and their corresponding grades. The CSV file will have two columns: "Student Name" and "Grade". The program should analyze the grades and provide the following functionalities:

1. Calculate the average grade for all students.
2. Determine the student with the highest grade.
3. Determine the student with the lowest grade.
4. Count the number of students who received a grade above a specified threshold.

**Solution:**

import csv

reader = csv.reader(file)

next(reader)

grades = {}

for row in reader:

student\_name, grade = row

if student\_name in grades:

grades[student\_name].append(int(grade))

else:

grades[student\_name] = [int(grade)]

for student\_name, student\_grades in grades.items():

average\_grade = sum(student\_grades) / len(student\_grades)

print(f'{student\_name}: Average Grade = {average\_grade}')

**Output:**

A white background with black text

Description automatically generated