1) Implement depth first search algorithm and Breadth First Search algorithm. Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

```
def create_graph():
    graph = {}
    vertices = int(input("Enter the number of vertices: "))

    for i in range(vertices):
        neighbors = list(map(int, input(f"Enter neighbors for vertex {i}: ").split()))
        graph[i] = neighbors

    return graph

def dfs(graph, vertex, visited):
    visited[vertex] = True
    print(vertex, end=' ')

    for neighbor in graph[vertex]:
        if not visited[neighbor]:
            dfs(graph, neighbor, visited)

def bfs(graph, start_vertex):
    visited = [False] * len(graph)
    queue = [start_vertex]  # Use a list as a queue
    visited[start_vertex] = True

    while queue:
        vertex = queue.pop(0)  # Dequeue from the front
        print(vertex, end=' ')

        for neighbor in graph[vertex]:
            if not visited[neighbor]:
                queue.append(neighbor)  # Enqueue to the back
                visited[neighbor] = True

# Create the graph
graph = create_graph()
start_vertex = int(input("Enter the starting vertex: "))

print("DFS:")
dfs(graph, start_vertex, [False] * len(graph))
print("\nBFS:")
bfs(graph, start_vertex)
```

OUTPUT:

```
Enter the number of vertices: 5
Enter neighbors for vertex 0: 1 2
Enter neighbors for vertex 1: 3 4
Enter neighbors for vertex 2: 0
Enter neighbors for vertex 3: 1
Enter neighbors for vertex 4: 1
Enter the starting vertex: 0
DFS:
0 1 3 4 2
BFS:
0 1 2 3 4
```

2) Develop an elementary chatbot for any suitable customer interaction application.

```
import random

# Greetings for the chatbot
greetings = ["Hello! How can I assist you today?", "Hi there! How can I help?", "Welcome! What can I do for you?"]

# Responses to common customer inquiries
responses = {
    "order_status": "To check your order status, please visit our website and log in to your account.",
    "product_info": "You can find detailed product information on our website. Do you have a specific product in mind?",
    "return_policy": "Our return policy allows returns within 30 days of purchase. Please review our website for more details.",
    "contact_info": "You can contact our customer support at support@example.com or call us at 123-456-7890.",
    "default": "I'm sorry, I couldn't understand your request. Please feel free to ask another question."
}

def chatbot_response(user_input):
    user_input = user_input.lower()

    if "hello" in user_input or "hi" in user_input:
        return random.choice(greetings)
    elif "order status" in user_input:
        return responses["order_status"]
    elif "product" in user_input:
        return responses["product_info"]
    elif "return policy" in user_input:
        return responses["return_policy"]
    elif "contact" in user_input or "support" in user_input:
        return responses["contact_info"]
    else:
        return responses["default"]

print("Customer Service Chatbot: Hello! How can I assist you today?")

while True:
    user_input = input("You: ")
    if user_input.lower() == "bye" or user_input.lower() == "exit":
        print("Customer Service Chatbot: Goodbye! Have a great day.")
        break
    response = chatbot_response(user_input)
    print("Customer Service Chatbot:", response)
```

OUTPUT:

Customer Service Chatbot: Hello! How can I assist you today?
You: hi
Customer Service Chatbot: Welcome! What can I do for you?
You: product details
Customer Service Chatbot: You can find detailed product information on our website. Do you have a specific product in mind?
You: bye
Customer Service Chatbot: Goodbye! Have a great day.

3) Implement Alpha-Beta Tree search for any game search problem.

```
MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        best = MIN
        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best

    else:
        best = MAX
        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best

if __name__ == "__main__":
    values = []
    for i in range(8):
        value = int(input(f"Enter value for node {i}: "))
        values.append(value)

    print("The optimal value is:", minimax(0, 0, True, values, MIN, MAX))
```

OUTPUT:

```
Enter value for node 0: 3
Enter value for node 1: 2
Enter value for node 2: 5
Enter value for node 3: 1
Enter value for node 4: -6
Enter value for node 5: -2
Enter value for node 6: 5
Enter value for node 7: 1
The optimal value is: 3
```

4) Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem

```
def is_safe(board, row, col):
    # Check if there is a queen in the same column
    for i in range(row):
        if board[i] == col or \
            board[i] - i == col - row or \
            board[i] + i == col + row:
             return False
    return True

def print_solution(board):
    print("-----------")
    for row in range(len(board)):
        line = ""
        for col in range(len(board)):
            line += " Q " if board[row] == col else " - "
        print(line)
    print("-----------\n")

def solve_n_queens_util(board, row):
    if row == len(board):
        print_solution(board)
        return

    for col in range(len(board)):
        if is_safe(board, row, col):
            board[row] = col
            solve_n_queens_util(board, row + 1)

def solve_n_queens(n):
    board = [-1] * n
    solve_n_queens_util(board, 0)

# Solve the 4x4 N-Queens problem
solve_n_queens(4)
```

OUTPUT:

```
-----------
 - Q - -
 - - - Q
 Q - - -
 - - Q -
-----------

-----------
 - - Q -
 Q - - -
 - - - Q
 - Q - -
-----------
```

5) Implement Greedy search algorithm for any of the following application: Selection Sort.

```
def greedy_selection_sort(arr):
    n = len(arr)

    for i in range(n):
        # Find the minimum element in the unsorted part of the array
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j

        # Swap the found minimum element with the first element
        arr[i], arr[min_index] = arr[min_index], arr[i]

    return arr

if __name__ == "__main__":
    # Take input from the user
    input_array = list(map(int, input("Enter space-separated numbers: ").split()))

    print("Input Array:", input_array)

    sorted_array = greedy_selection_sort(input_array.copy())
    print("Sorted Array:", sorted_array)
```

OUTPUT:

```
Enter space-separated numbers: 21 63 54 76 20 16 49 76 32
Input Array: [21, 63, 54, 76, 20, 16, 49, 76, 32]
Sorted Array: [16, 20, 21, 32, 49, 54, 63, 76, 76]
```

1) Borrower.

```
create database library;
use library;
create table borrower (Roll_no int(10), Name varchar(20), Date_of_Issue date, Name_of_Book varchar(20), Status
varchar(2)) ;
create table fine (Roll_no int(10), Date_ date, Amt int(8)) ;

insert into borrower values(1,'abc','2023-08-25','SEPM','I');
insert into borrower values(2,'xyz','2023-09-01','AI','I');
insert into borrower values(3,'pqr','2023-08-15','DBMS','I');

delimiter $

create procedure fine_calculator(in rollno int(10))
 begin
 declare
  fine1 int ;
 declare
  DOI date;
 declare
  no_of_days int;
 declare
  exit handler for SQLEXCEPTION select'create table definition';
  select  Date_of_Issue into DOI from borrower where Roll_no=rollno ;
  select  datediff(curdate(),DOI) into no_of_days;

  if no_of_days>15 and no_of_days<=30 then set fine1= (no_of_days-15) * 5;
   insert into fine values(rollno,curdate(),fine1);

  elseif no_of_days>30 then set fine1=(no_of_days-30)*50 + 15*5;
   insert into fine values (rollno,curdate(),fine1);

  else
   insert into fine values(rollno,curdate(),0);

  end if;

  update borrower set Status='R' where Roll_no=rollno;
end $

delimiter ;

call fine_calculator(1);
call fine_calculator(2);
call fine_calculator(3);

select "⇓⇓⇓ Following is the borrower table ⇓⇓⇓" as "";
select * from borrower;
select "⇓⇓⇓ Following is the fine table ⇓⇓⇓" as "";
select * from fine;

drop database Library;
```

2) Area.

```
create database area;
use area;
create table calc(length int,breadth int,area int);


DELIMITER $

CREATE PROCEDURE calculate_area_and_perimeter()
BEGIN
  DECLARE v_length INT;
  DECLARE v_breadth INT;
  DECLARE v_areaf INT;
  DECLARE v_perimeter INT;


  SET v_breadth := 10;
  SET v_length := 10;

  WHILE v_length <= 20 DO

    SET v_areaf := v_length * v_breadth;
    SET v_perimeter := 2 * (v_length + v_breadth);


    INSERT INTO calc VALUES (v_length, v_breadth, v_areaf);

    SELECT CONCAT('Length: ', v_length, ', Breadth: ', v_breadth, ', Area: ', v_areaf, ', Perimeter: ', v_perimeter);


    SET v_length := v_length + 1;
  END WHILE;
END $


DELIMITER ;


CALL calculate_area_and_perimeter();
```

3) Salary.

```
create database Company;
use Company;
create table Employee (Empid int(10) primary key, Name varchar(20), Basic_salary int(10), type varchar(20));
create table Salary (Empid int(10) ,basic_salary int(10),gross_salary int(10), net_salary int(10) );

insert into Employee values(101,"parimal",20000,"Permanent");
insert into Employee values(102,"Saish",21000,"Permanent");

delimiter $

create procedure salary_count(in emp_id int(10))
begin
 declare emp_type varchar(15);
 declare Basic_salary int(10);
 declare DA int(10);
 declare HRA int(10);
 declare net_salary int(10);
 declare income_tax int(10);
 declare Gross_Salary int(10);
 declare Deduction int(10);
 declare
 exit handler for SQLEXCEPTION select'create table definition';
 select E.Basic_salary into Basic_salary from Employee E where E.Empid=emp_id;
 select type into emp_type from Employee where Empid=emp_id;
 if emp_type="Permanent" then set Deduction=2000 ;
  set DA=(Basic_salary)*0.50;
  set HRA=(Basic_salary)*0.12;
  set Gross_salary=Basic_salary+DA+HRA;
  set net_salary=Gross_salary-Deduction;

 insert into Salary values(emp_id,Basic_salary, Gross_salary,net_salary) ;
 end if;
end $
delimiter ;

call salary_count(101);
call salary_count(102);

select * from Employee;
select * from Salary ;

drop database Company;
```