Assignment - 1A

Title - DFS and BFS using recursive algorithm

DFS

Program in C++

```
// C++ program to print DFS traversal from
// a given vertex in a  given graph
#include &lt;bits/stdc++.h&gt;
using namespace std;

// Graph class represents a directed graph
// using adjacency list representation
class Graph {
public:
   map&lt;int, bool&gt; visited;
   map&lt;int, list&lt;int&gt; &gt; adj;

   // function to add an edge to graph
   void addEdge(int v, int w);

   // DFS traversal of the vertices
   // reachable from v
   void DFS(int v);
};

void Graph::addEdge(int v, int w)

{
   adj[v].push_back(w); // Add w to v’s list.
}

void Graph::DFS(int v)
{
   // Mark the current node as visited and
   // print it
   visited[v] = true;
   cout &lt;&lt; v &lt;&lt; &quot; &quot;;

   // Recur for all the vertices adjacent
   // to this vertex
   list&lt;int&gt;::iterator i;
```

```
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFS(*i);
}

// Driver&#39;s code
int main()
{
    // Create a graph given in the above diagram
    Graph g;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout &lt;&lt; &quot;Following is Depth First Traversal&quot;
            &quot; (starting from vertex 2) \n&quot;;

    // Function call
    g.DFS(2);

    return 0;
}
```

Output
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
Time complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph.

Assignment - 1B

BFS

Program in C++

```
// C++ code to print BFS traversal from a given
// source vertex

#include &lt;bits/stdc++.h&gt;
using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph {

   // No. of vertices
   int V;

   // Pointer to an array containing adjacency lists
   vector&lt;list&lt;int&gt; &gt; adj;

public:
   // Constructor
   Graph(int V);

   // Function to add an edge to graph
   void addEdge(int v, int w);

   // Prints BFS traversal from a given source s
   void BFS(int s);
};


Graph::Graph(int V)
{
   this-&gt;V = V;
   adj.resize(V);
}

void Graph::addEdge(int v, int w)
```

```
{
    // Add w to v’s list.
    adj[v].push_back(w);
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    vector&lt;bool&gt; visited;
    visited.resize(V, false);

    // Create a queue for BFS
    list&lt;int&gt; queue;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);

    while (!queue.empty()) {


        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout &lt;&lt; s &lt;&lt; &quot; &quot;;
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (auto adjacent : adj[s]) {
            if (!visited[adjacent]) {
                visited[adjacent] = true;
                queue.push_back(adjacent);
            }
        }
    }
}

// Driver code
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
```

```
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout &lt;&lt; &quot;Following is Breadth First Traversal &quot;
         &lt;&lt; &quot;(starting from vertex 2) \n&quot;;
    g.BFS(2);

    return 0;
}
```

Output

Following is Breadth First Traversal (starting from vertex 2)

2 0 3 1

Assignment - 3

Tile - A* algorithm for 8 puzzle problem

A* Algorithm

Code

```
// Program to print path from root node to destination node
// for N*N -1 puzzle algorithm using Branch and Bound
// The solution assumes that instance of puzzle is solvable
#include <bits/stdc++.h>
using namespace std;
#define N 3

// state space tree nodes
struct Node
{
	// stores the parent node of the current node
	// helps in tracing path when the answer is found
	Node* parent;

	// stores matrix
	int mat[N][N];

	// stores blank tile coordinates
	int x, y;

	// stores the number of misplaced tiles
	int cost;

	// stores the number of moves so far
	int level;
};

// Function to print N x N matrix
int printMatrix(int mat[N][N])
{
	for (int i = 0; i < N; i++)
	{
```

```
		for (int j = 0; j < N; j++)
			printf("%d ", mat[i][j]);
		printf("\n");
	}
}

// Function to allocate a new node
Node* newNode(int mat[N][N], int x, int y, int newX,
			int newY, int level, Node* parent)
{
	Node* node = new Node;

	// set pointer for path to root
	node->parent = parent;

	// copy data from parent node to current node
	memcpy(node->mat, mat, sizeof node->mat);

	// move tile by 1 position
	swap(node->mat[x][y], node->mat[newX][newY]);

	// set number of misplaced tiles
	node->cost = INT_MAX;

	// set number of moves so far
	node->level = level;

	// update new blank tile coordinates
	node->x = newX;
	node->y = newY;

	return node;
}

// bottom, left, top, right
int row[] = { 1, 0, -1, 0 };
int col[] = { 0, -1, 0, 1 };

// Function to calculate the number of misplaced tiles
// ie. number of non-blank tiles not in their goal position
int calculateCost(int initial[N][N], int final[N][N])
{
	int count = 0;
	for (int i = 0; i < N; i++)
```

```
	for (int j = 0; j < N; j++)
		if (initial[i][j] && initial[i][j] != final[i][j])
		count++;
	return count;
}

// Function to check if (x, y) is a valid matrix coordinate
int isSafe(int x, int y)
{
	return (x >= 0 && x < N && y >= 0 && y < N);
}

// print path from root node to destination node
void printPath(Node* root)
{
	if (root == NULL)
		return;
	printPath(root->parent);
	printMatrix(root->mat);

	printf("\n");
}

// Comparison object to be used to order the heap
struct comp
{
	bool operator()(const Node* lhs, const Node* rhs) const
	{
		return (lhs->cost + lhs->level) > (rhs->cost + rhs->level);
	}
};

// Function to solve N*N - 1 puzzle algorithm using
// Branch and Bound. x and y are blank tile coordinates
// in initial state
void solve(int initial[N][N], int x, int y,
		int final[N][N])
{
	// Create a priority queue to store live nodes of
	// search tree;
	priority_queue<Node*, std::vector<Node*>, comp> pq;

	// create a root node and calculate its cost
	Node* root = newNode(initial, x, y, x, y, 0, NULL);
```

```
root->cost = calculateCost(initial, final);

// Add root to list of live nodes;
pq.push(root);

// Finds a live node with least cost,
// add its childrens to list of live nodes and
// finally deletes it from the list.
while (!pq.empty())
{
	// Find a live node with least estimated cost
	Node* min = pq.top();

	// The found node is deleted from the list of
	// live nodes
	pq.pop();

	// if min is an answer node
	if (min->cost == 0)
	{
		// print the path from root to destination;
		printPath(min);
		return;
	}

	// do for each child of min
	// max 4 children for a node
	for (int i = 0; i < 4; i++)
	{
		if (isSafe(min->x + row[i], min->y + col[i]))
		{
			// create a child node and calculate
			// its cost
			Node* child = newNode(min->mat, min->x,
						min->y, min->x + row[i],
						min->y + col[i],
						min->level + 1, min);
			child->cost = calculateCost(child->mat, final);

			// Add child to list of live nodes
			pq.push(child);
		}
	}
}
```

```
}

// Driver code
int main()
{
	// Initial configuration
	// Value 0 is used for empty space
	int initial[N][N] =
	{
		{1, 2, 3},
		{5, 6, 0},
		{7, 8, 4}
	};

	// Solvable Final configuration
	// Value 0 is used for empty space
	int final[N][N] =
	{
		{1, 2, 3},
		{5, 8, 6},
		{0, 7, 4}
	};

	// Blank tile coordinates in initial
	// configuration
	int x = 1, y = 2;

	solve(initial, x, y, final);

	return 0;
}
```

Output

```
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4
```

```
1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4
```

Assignment - 5

Title - Implement Greedy search algorithm

Selection sort Algorithm

Code
```
// C++ program for implementation of
// selection sort
#include <bits/stdc++.h>
using namespace std;

//Swap function
void swap(int *xp, int *yp)
{
	int temp = *xp;
	*xp = *yp;
	*yp = temp;
}

void selectionSort(int arr[], int n)
{
	int i, j, min_idx;
	// One by one move boundary of
	// unsorted subarray
	for (i = 0; i < n-1; i++)
	{
		// Find the minimum element in
		// unsorted array
		min_idx = i;
		for (j = i+1; j < n; j++)
		{
		if (arr[j] < arr[min_idx])
			min_idx = j;
		}
		// Swap the found minimum element
		// with the first element
		if (min_idx!=i)
			swap(&arr[min_idx], &arr[i]);
	}
}

//Function to print an array
void printArray(int arr[], int size)
```

```
{
	int i;
	for (i=0; i < size; i++)
	{
	cout << arr[i] << " ";
	cout << endl;
	}
}

// Driver program to test above functions
int main()
{
	int arr[] = {64, 25, 12, 22, 11};
	int n = sizeof(arr)/sizeof(arr[0]);
	selectionSort(arr, n);
	cout << "Sorted array: \n";
	printArray(arr, n);
	return 0;
}
\
```

Output

```
Sorted array:
11 12 22 25 64
```

Assignment - 7

Title - CSP using DFS N-queens problem

Code

```
# Python3 program to solve N Queen
# Problem using backtracking
global N
N = 4

def printSolution(board):
	for i in range(N):
		for j in range(N):
			print(board[i][j], end = " ")
		print()

def isSafe(board, row, col):

	# Check this row on left side
	for i in range(col):
		if board[row][i] == 1:
			return False

	# Check upper diagonal on left side
	for i, j in zip(range(row, -1, -1),
					range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	# Check lower diagonal on left side
	for i, j in zip(range(row, N, 1),
					range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	return True

def solveNQUtil(board, col):

	# base case: If all queens are placed
	# then return true
	if col >= N:
		return True
```

```
    for i in range(N):

        if isSafe(board, i, col):

            # Place this queen in board[i][col]
            board[i][col] = 1

            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True

            # If placing queen in board[i][col
            # doesn't lead to a solution, then
            # queen from board[i][col]
            board[i][col] = 0

    return False

def solveNQ():
    board = [ [0, 0, 0, 0],
            [0, 0, 0, 0],
            [0, 0, 0, 0],
            [0, 0, 0, 0] ]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver Code
solveNQ()
```

Output

```
. . Q .
Q . . .
. . . Q
. Q . .
```

Assignment - 9

Title - Chatbot Application

Chatbot program:

```
def greet(bot_name, birth_year):
print("Hello! My name is {0}.".format(bot_name))
print("I was created in {0}.".format(birth_year))
def remind_name():
print('Please, remind me your name.')
name = input()
print("What a great name you have, {0}!".format(name))

def guess_age():
print('Let me guess your age.')
print('Enter remainders of dividing your age by 3, 5 and 7.')
rem3 = int(input())
rem5 = int(input())
rem7 = int(input())
age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105
print("Your age is {0}; that's a good time to start
programming!".format(age))
def count():
print('Now I will prove to you that I can count to any number you
want.')
num = int(input())
counter = 0
while counter <= num:
print("{0} !".format(counter))
counter += 1

def test():
print("Let's test your programming knowledge.")
print("Why do we use methods?")
print("1. To repeat a statement multiple times.")
print("2. To decompose a program into several small subroutines.")
print("3. To determine the execution time of a program.")
print("4. To interrupt the execution of a program.")
answer = 2
guess = int(input())
while guess != answer:
print("Please, try again.")
guess = int(input())
```

```
print('Completed, have a nice day!')
print('.................................')
print('.................................')
print('.................................')
def end():
print('Congratulations, have a nice day!')
print('.................................')
print('.................................')
print('.................................')
input()
greet('Sbot', '2021') # change it as you need
remind_name()
guess_age()
count()
test()
end()
```

Output:
Hello! My name is Sbot.
I was created in 2021.
Please, remind me your name.

Assignment - 11

Title - Expert system

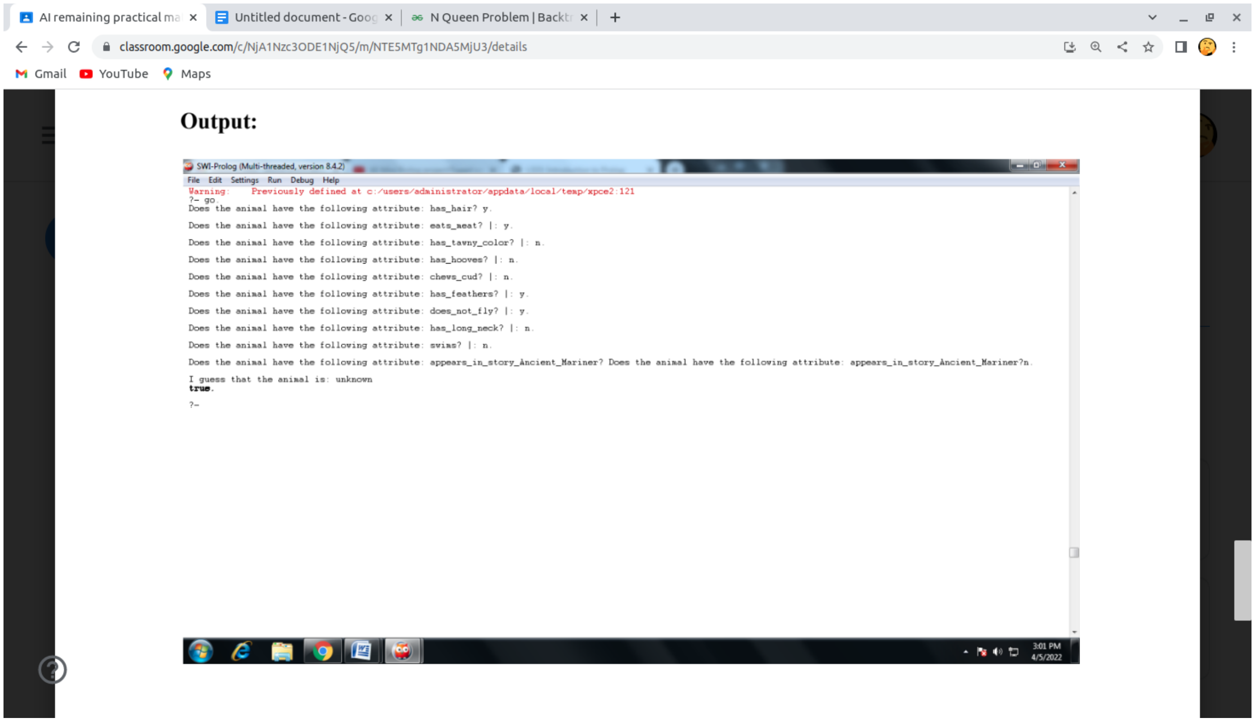
program: The animal identification game (simple expert system)

```
go :- hypothesize(Animal),
write('I guess that the animal is: '),
write(Animal),
nl,
undo.
/* hypotheses to be tested */
hypothesize(cheetah) :- cheetah, !.
hypothesize(tiger) :- tiger, !.

hypothesize(giraffe) :- giraffe, !.
hypothesize(zebra) :- zebra, !.
hypothesize(ostrich) :- ostrich, !.
hypothesize(penguin) :- penguin, !.
hypothesize(albatross) :- albatross, !.
hypothesize(unknown). /* no diagnosis */
/* animal identification rules */
cheetah :- mammal,
carnivore,
verify(has_tawny_color),
verify(has_dark_spots).
tiger :- mammal,
carnivore,
verify(has_tawny_color),
verify(has_black_stripes).
giraffe :- ungulate,
verify(has_long_neck),
verify(has_long_legs).
zebra :- ungulate,
verify(has_black_stripes).
ostrich :- bird,
verify(does_not_fly),
verify(has_long_neck).
penguin :- bird,
verify(does_not_fly),
verify(swims),
verify(is_black_and_white).
albatross :- bird,
```

```
verify(appears_in_story_Ancient_Mariner),
verify(flys_well).
/* classification rules */
mammal :- verify(has_hair), !.
mammal :- verify(gives_milk).
bird :- verify(has_feathers), !.
bird :- verify(flys),
verify(lays_eggs).
carnivore :- verify(eats_meat), !.
carnivore :- verify(has_pointed_teeth),
verify(has_claws),
verify(has_forward_eyes).
ungulate :- mammal,
verify(has_hooves), !.
ungulate :- mammal,
verify(chews_cud).
/* how to ask questions */
ask(Question) :-
write('Does the animal have the following attribute: '),
write(Question),
write('? '),
read(Response),
( (Response == yes ; Response == y)
->
assert(yes(Question)) ;
assert(no(Question)), fail).
:- dynamic yes/1,no/1.
/* How to verify something */
verify(S) :-
(yes(S)
->
true ;
(no(S)
->
fail ;
ask(S))).
/* undo all yes/no assertions */
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

Output:



Output:
SWI-Prolog (Multi-threaded, version 8.4.2)
File Edit Settings Run Debug Help
Warning: Previously defined at c:/users/administrator/appdata/local/temp/xpce2:121
?- go.
Does the animal have the following attribute: has_hair? y.
Does the animal have the following attribute: eats_meat? |: y.
Does the animal have the following attribute: has_tawny_color? |: n.
Does the animal have the following attribute: has_hooves? |: n.
Does the animal have the following attribute: chews_cud? |: n.
Does the animal have the following attribute: has_feathers? |: y.
Does the animal have the following attribute: does_not_fly? |: y.
Does the animal have the following attribute: has_long_neck? |: n.
Does the animal have the following attribute: swims? |: n.
Does the animal have the following attribute: appears_in_story_Ancient_Mariner? Does the animal have the following attribute: appears_in_story_Ancient_Mariner?n.
I guess that the animal is: unknown
true.
?-