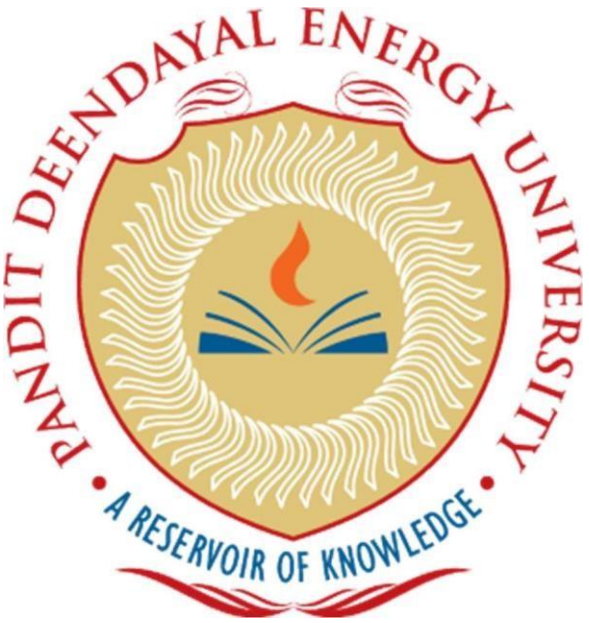
PANDIT DEENDAYAL ENERGY UNIVERSITY
COMPUTER ENGINEERING



Artificial Intelligence
Lab Manual

Submitted To:
Dr. Rajeev Gupta

Submitted By:
Bhut Tushar G.
20BCP023

WATER JUG PROBLEM [DFS]

NAME : BHUT TUSHAR G.
ROLL NO : 20BCP023

```cpp
/*
    Name : Bhut Tushar
*/
#include <bits/stdc++.h>

#define ll long long
#define ull unsigned long long
#define sz size()
#define vll vector<ll>
#define mp make_pair
#define pb push_back
#define ppb pop_back
#define fi first
#define se second
#define no cout << "NO" << endl;
#define yes cout << "YES" << endl;
#define mod 1000000007
#define all(x) (x).begin(), (x).end()
#define SORT(v) sort(all(v))
#define REVSORT(v) sort(all(v), greater<int>())
#define MAX(v) max_element(all(v))
#define MIN(v) min_element(all(v))
#define rep(from, to) for (int i = from; i <= to; i++)
#define rep_back(from, to) for (int i = from; i >= to; i--)
#define take(v) rep(0, v.size()) cin >> v[i];
#define FASTIO                          \
    ios_base::sync_with_stdio(false); \
    cin.tie(NULL);                    \
    cout.tie(NULL)
using namespace std;

/* ========== YOUR CODE HERE ========== */

vector<pair<pair<int,int>,pair<int, int>>> traversal_path;

bool dfs(int jug1, int jug2, int curr1, int curr2, pair<int, int> goal, map<pair<int, int>, bool> &visited)
{
    if (!visited[{curr1,curr2}] and (mp(curr1, curr2) == goal))
    {
        return true;
    }
    if (visited[{curr1, curr2}])
        return false;

    visited[{curr1, curr2}] = true;

    // fill
    if (curr1 < jug1 and !visited[{jug1, curr2}] and dfs(jug1, jug2, jug1, curr2, goal, visited))
    {
        traversal_path.push_back({{curr1,curr2},{jug1, curr2}});
        return true;
    }
    if (curr2 < jug2 and !visited[{curr1, jug2}] and dfs(jug1, jug2, curr1, jug2, goal, visited))
    {
```

```
        traversal_path.push_back({{curr1,curr2},{curr1, jug2}});
        return true;
    }

    // empty
    if (curr1 > 0 and !visited[{0, curr2}] and dfs(jug1, jug2, 0, curr2, goal, visited))
    {
        traversal_path.push_back({{curr1,curr2},{0, curr2}});
        return true;
    }
    if (curr2 > 0 and !visited[{curr1, 0}] and dfs(jug1, jug2, curr1, 0, goal, visited))
    {
        traversal_path.push_back({{curr1,curr2},{curr1, 0}});
        return true;
    }

    // transfer
    if (curr1 > 0 and curr2 < jug2)
    {
        if (curr1 + curr2 <= jug2 and !visited[{0, curr1 + curr2}] and dfs(jug1, jug2, 0, curr1 + curr2, goal, visited))
        {
            traversal_path.push_back({{curr1,curr2},{0, curr1 + curr2}});
            return true;
        }
        else if (!visited[{curr1 + curr2 - jug2, jug2}] and dfs(jug1, jug2, curr1 + curr2 - jug2, jug2, goal, visited))
        {
            traversal_path.push_back({{curr1,curr2},{curr1 + curr2 - jug2, jug2}});
            return true;
        }
    }
    if (curr2 > 0 and curr1 < jug1)
    {
        if (curr1 + curr2 <= jug1 and !visited[{curr1 + curr2, 0}] and dfs(jug1, jug2, curr1 + curr2, 0, goal, visited))
        {
            traversal_path.push_back({{curr1,curr2},{curr1 + curr2, 0}});
            return true;
        }
        else if (!visited[{jug1, curr1 + curr2 - jug1}] and dfs(jug1, jug2, jug1, curr1 + curr2 - jug1, goal, visited))
        {
            traversal_path.push_back({{curr1,curr2},{jug1, curr1 + curr2 - jug1}});
            return true;
        }
    }
    return false;
}
int main()
{
    FASTIO;
    int jug1 = 4;
    int jug2 = 3;
    pair<int, int> goal = {4, 2};
    map<pair<int, int>, bool> visited;
    visited[{0,0}] = false;
```

```
    if (dfs(jug1, jug2, 0, 0, goal, visited))
    {
        cout << "\n*** Solution Exist ***\n";
        reverse(all(traversal_path));

        pair<int,int>node = goal;

        for(auto i:traversal_path)
        {
            cout<<i.first.first<<" "<<i.first.second<<"  ->  "<<i.second.first<<" "<<i.second.second<<"\n";
        }
    }
    else
    {
        cout << "\n*** Solution Not Exist ***\n";
    }
    return 0;
}
```

Output : Goal = [2,0]

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CO
PS E:\My-Programs> cd "e:\My-Programs\" ;

*** Solution Exist ***
0 0  ->  4 0
4 0  ->  4 3
4 3  ->  0 3
0 3  ->  3 0
3 0  ->  3 3
3 3  ->  4 2
4 2  ->  0 2
0 2  ->  2 0
PS E:\My-Programs>
```

Output : Goal = [2,2]

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE
PS E:\My-Programs> cd "e:\My-Programs\" ; if ($

*** Solution Not Exist ***
PS E:\My-Programs>
```

WATER JUG PROBLEM [BFS]

NAME : BHUT TUSHAR G.
ROLL NO : 20BCP023

Code :

```cpp
/*
   ॐ श्री गणेशाय नमः
   ॐ नमः शिवाय
   Name : Bhut Tushar
*/

#include <bits/stdc++.h>

#define ll long long
#define ull unsigned long long
#define sz size()
#define vll vector<ll>
#define mp make_pair
#define pb push_back
#define ppb pop_back
#define fi first
#define se second
#define no cout << "NO" << endl;
#define yes cout << "YES" << endl;
#define mod 1000000007
#define all(x) (x).begin(), (x).end()
#define SORT(v) sort(all(v))
#define REVSORT(v) sort(all(v), greater<int>())
#define MAX(v) max_element(all(v))
#define MIN(v) min_element(all(v))
#define rep(from, to) for (int i = from; i <= to; i++)
#define rep_back(from, to) for (int i = from; i >= to; i--)
#define take(v) rep(0, v.size()) cin >> v[i];
#define FASTIO                      \
    ios_base::sync_with_stdio(false); \
    cin.tie(NULL);                  \
    cout.tie(NULL)
using namespace std;

/* ========== YOUR CODE HERE ========= */

vector<pair<pair<int, int>, pair<int, int>>> traversal_path;
map<pair<int, int>, set<pair<int, int>>> mp;

bool bfs(int jug1, int jug2, pair<int, int> goal)
{
    // visited[{0, 0}] = true;
    map<pair<int, int>, bool> visited;
    queue<pair<int, int>> q;
    q.push({0, 0});
```

```
while (!q.empty())
{
    pair<int, int> top = q.front();
    q.pop();

    if (top == goal)
    {
        return true;
    }

    if (!visited[top])
    {
        visited[top] = true;

        // fill
        if (top.first < jug1)
        {
            if (!visited[{jug1, top.second}])
            {
                q.push({jug1, top.second});
                traversal_path.push_back({top, {jug1, top.second}});
            }
        }
        if (top.second < jug2)
        {
            if (!visited[{top.first, jug2}])
            {
                q.push({top.first, jug2});
                traversal_path.push_back({top, {top.first, jug2}});
            }
        }

        // empty
        if (top.first > 0)
        {
            if (!visited[{0, top.second}])
            {
                q.push({0, top.second});
                traversal_path.push_back({top, {0, top.second}});
            }
        }
        if (top.second > 0)
        {
            if (!visited[{top.first, 0}])
            {
                q.push({top.first, 0});
                traversal_path.push_back({top, {top.first, 0}});
            }
        }
```

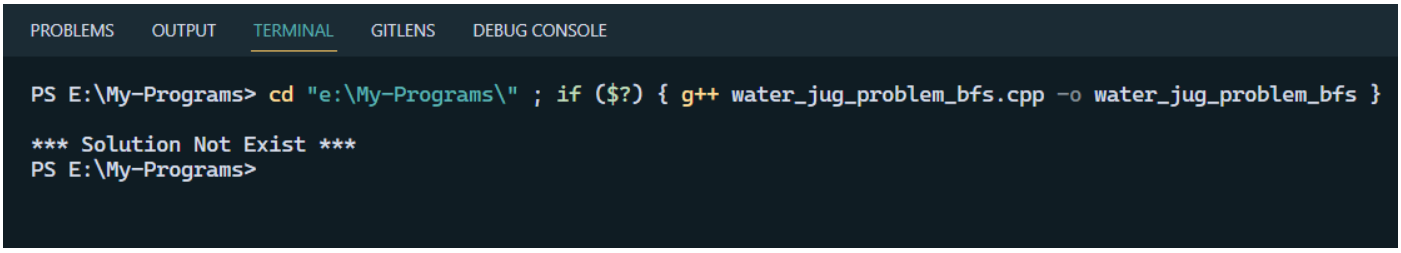
```
        // transfer
        if (top.first > 0 and top.second < jug2)
        {
            if (top.first + top.second <= jug2)
            {
                if (!visited[{0, top.first + top.second}])
                {
                    q.push({0, top.first + top.second});
                    traversal_path.push_back({top, {0, top.first + top.second}});
                }
            }
            else
            {
                if (!visited[{top.first + top.second - jug2, jug2}])
                {
                    q.push({top.first + top.second - jug2, jug2});
                    traversal_path.push_back({top, {top.first + top.second - jug2, jug2}});
                }
            }
        }
        if (top.second > 0 and top.first < jug1)
        {
            if (top.first + top.second <= jug1)
            {
                if (!visited[{top.first + top.second, 0}])
                {
                    q.push({top.first + top.second, 0});
                    traversal_path.push_back({top, {top.first + top.second, 0}});
                }
            }
            else
            {
                if (!visited[{jug1, top.first + top.second - jug1}])
                {
                    q.push({jug1, top.first + top.second - jug1});
                    traversal_path.push_back({top, {jug1, top.first + top.second - jug1}});
                }
            }
        }
    }
    // if solution not exists
    return false;
}




int main()
{
```

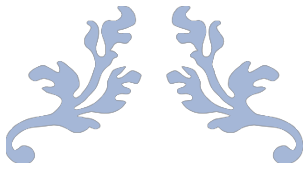
```
    FASTIO;

    int jug1 = 4;
    int jug2 = 3;
    pair<int, int> goal = {2, 0};

    if (bfs(jug1, jug2, goal))
    {
        cout << "\n*** Solution Exist ***\n";

        pair<int, int> node = goal;

        reverse(all(traversal_path));
        vector<pair<int, int>> path;
        path.push_back(goal);
        for (auto i : traversal_path)
        {
            if (i.second == node)
            {
                node = i.first;
                path.push_back(node);
            }
        }
        reverse(all(path));
        for (auto i : path)
            cout << i.first << " " << i.second << "\n";
    }
    else
        cout << "\n*** Solution Not Exist ***\n";

    return 0;
}
```

Output : Goal = [2,0]

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS E:\My-Programs> cd "e:\My-Programs\" ; if ($?) { g++ water_jug_problem_bfs.cpp -o water_jug_problem_bfs }

*** Solution Exist ***
0 0
4 0
1 3
1 0
0 1
4 1
2 3
2 0
PS E:\My-Programs>
```

Goal = [2,2]


PROBLEMS
OUTPUT
TERMINAL
GITLENS
DEBUG CONSOLE
PS E:\My-Programs> cd "e:\My-Programs\" ; if ($?) { g++ water_jug_problem_bfs.cpp -o water_jug_problem_bfs }
*** Solution Not Exist ***
PS E:\My-Programs>

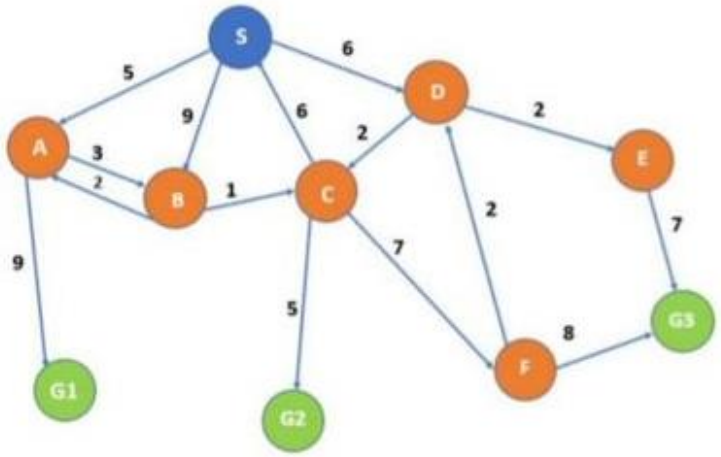


UNIFORM COST SEARCH

NAME : BHUT TUSHAR G.
ROLL NO : 20BCP023

Assignment-3

Implement uniform cost search and find the path from S-G2 for a given graph.



Code :

```
#include <bits/stdc++.h>

#define ll long long
#define ull unsigned long long
#define sz size()
#define vll vector<ll>
#define mp make_pair
#define pb push_back
#define ppb pop_back
#define fi first
#define se second
#define no cout << "NO" << endl;
#define yes cout << "YES" << endl;
#define mod 1000000007
#define all(x) (x).begin(), (x).end()
#define SORT(v) sort(all(v))
#define REVSORT(v) sort(all(v), greater<int>())
#define MAX(v) max_element(all(v))
#define MIN(v) min_element(all(v))
#define rep(from, to) for (int i = from; i <= to; i++)
#define rep_back(from, to) for (int i = from; i >= to; i--)
#define take(v) rep(0, v.size()) cin >> v[i];
#define FASTIO                      \
    ios_base::sync_with_stdio(false); \
    cin.tie(NULL);                  \
    cout.tie(NULL)
using namespace std;
```

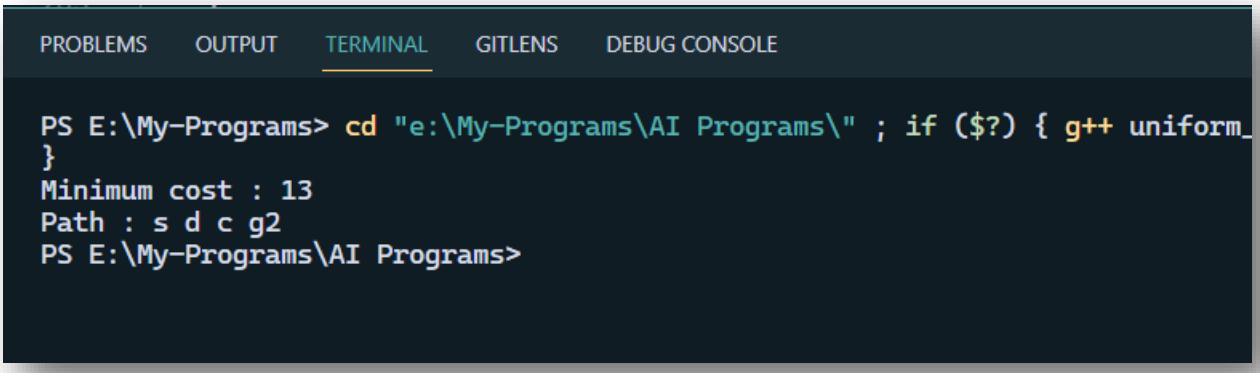
```cpp
/* ========== YOUR CODE HERE ========== */
struct ComparePairs
{
    bool operator()(const pair<int, string> &a, const pair<int, string> &b)
    {
        return a.first > b.first; // compares the integers (first element of the pair)
    }
};

map<string, string> parent;

void bfs(map<string, vector<string>> &edges, string source, map<pair<string, string>, int> &cost,string goal)
{
    priority_queue<pair<int, string>, vector<pair<int, string>>, ComparePairs> pq;

    map<string, bool> visited;
    pq.push({0, source});

    map<string,int>costOfAllNode;
    for(auto i:edges)
    {
        costOfAllNode[i.first] = INT_MAX;
    }
    // parent[source] = source;

    while (!pq.empty())
    {
        pair<int, string> top = pq.top();
        pq.pop();

        // cout<<top.second<<"\n";
        if (top.second == goal)
        {
            // parent["g3"] = top.second;
            cout << "Minimum cost : " << top.first << "\n";
            return;
        }

        visited[top.second] = true;

        for (auto i : edges[top.second])
        {
            if (!visited[i] and costOfAllNode[i] > costOfAllNode[top.second]+cost[{top.second,i}])
            {
                parent[i] = top.second;
                costOfAllNode[i] = costOfAllNode[top.second]+cost[{top.second,i}];
                pq.push({top.first + cost[{top.second, i}], i});
            }
        }
    }
}
int main()
{
    FASTIO;
```

```
map<string, vector<string>> edges;

// directed Graph
edges["a"].push_back("b");
edges["b"].push_back("a");
edges["a"].push_back("g1");
edges["s"].push_back("a");
edges["s"].push_back("b");
edges["s"].push_back("d");
edges["b"].push_back("c");
edges["c"].push_back("g2");
edges["c"].push_back("f");
edges["c"].push_back("s");
edges["d"].push_back("s");
edges["d"].push_back("c");
edges["d"].push_back("e");
edges["f"].push_back("g3");
edges["f"].push_back("d");
edges["e"].push_back("g3");

//edges-cost
map<pair<string, string>, int> cost;

cost[{"a", "b"}] = 3;
cost[{"a", "g1"}] = 9;
cost[{"b", "a"}] = 2;
cost[{"b", "c"}] = 1;
cost[{"s", "a"}] = 5;
cost[{"s", "b"}] = 9;
cost[{"s", "d"}] = 6;
cost[{"c", "s"}] = 6;
cost[{"c", "g2"}] = 5;
cost[{"c", "f"}] = 7;
cost[{"d", "s"}] = 1;
cost[{"d", "e"}] = 2;
cost[{"d", "c"}] = 2;
cost[{"e", "g3"}] = 7;
cost[{"f", "g3"}] = 8;
cost[{"f", "d"}] = 2;


//Undirected Graph

// edges["a"].push_back("b");
// edges["b"].push_back("a");
// edges["a"].push_back("c");
// edges["c"].push_back("a");
// edges["a"].push_back("d");
// edges["d"].push_back("a");
// edges["b"].push_back("d");
// edges["d"].push_back("b");
// edges["b"].push_back("e");
// edges["e"].push_back("b");
```

```
// edges["c"].push_back("d");
// edges["d"].push_back("c");
// edges["c"].push_back("g1");
// edges["g1"].push_back("c");
// edges["d"].push_back("e");
// edges["e"].push_back("d");
// edges["d"].push_back("f");
// edges["f"].push_back("d");
// edges["d"].push_back("g1");
// edges["g1"].push_back("d");
// edges["e"].push_back("f");
// edges["f"].push_back("e");
// edges["e"].push_back("g2");
// edges["g2"].push_back("e");
// edges["f"].push_back("g1");
// edges["g1"].push_back("f");
// edges["f"].push_back("g2");
// edges["g2"].push_back("f");
// edges["f"].push_back("g3");
// edges["g3"].push_back("f");

// cost[{"a", "b"}] = 1;
// cost[{"b", "a"}] = 1;
// cost[{"a", "c"}] = 3;
// cost[{"c", "a"}] = 3;
// cost[{"a", "d"}] = 2;
// cost[{"d", "a"}] = 2;
// cost[{"b", "d"}] = 4;
// cost[{"d", "b"}] = 4;
// cost[{"b", "e"}] = 1;
// cost[{"e", "b"}] = 1;
// cost[{"c", "d"}] = 2;
// cost[{"d", "c"}] = 2;
// cost[{"c", "g1"}] = 3;
// cost[{"g1", "c"}] = 3;
// cost[{"d", "e"}] = 1;
// cost[{"e", "d"}] = 1;
// cost[{"d", "f"}] = 1;
// cost[{"f", "d"}] = 1;
// cost[{"d", "g1"}] = 5;
// cost[{"g1", "d"}] = 5;
// cost[{"e", "g2"}] = 2;
// cost[{"g2", "e"}] = 2;
// cost[{"e", "f"}] = 3;
// cost[{"f", "e"}] = 3;
// cost[{"f", "g1"}] = 2;
// cost[{"g1", "f"}] = 2;
// cost[{"f", "g2"}] = 2;
// cost[{"g2", "f"}] = 2;
// cost[{"f", "g3"}] = 2;
// cost[{"g3", "f"}] = 2;

string source = "s";
string goal = "g2";
```

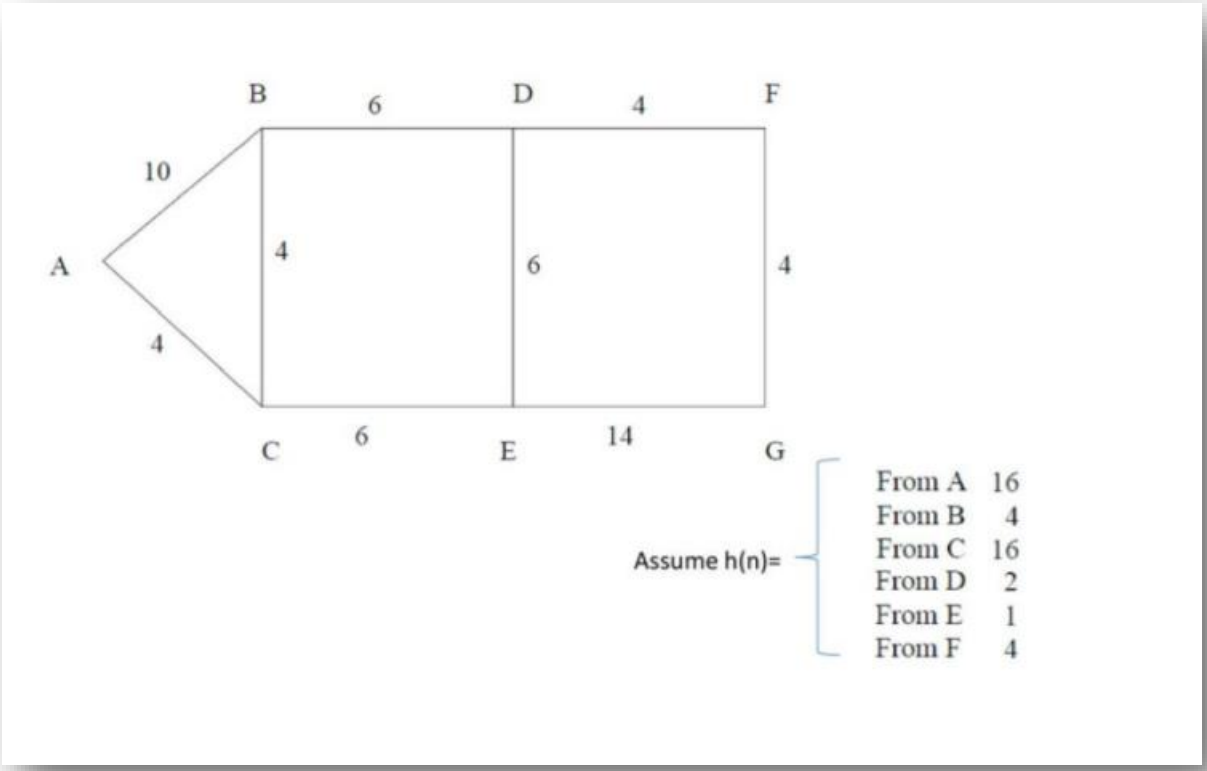
```
    bfs(edges, source, cost,goal);
    cout<<"Path : ";

    vector<string> path;
    while (goal != source)
    {
        path.push_back(goal);
        goal = parent[goal];
    }
    path.push_back(source);

    reverse(all(path));
    for(auto i:path)
    {
        cout<<i<<" ";
    }

    return 0;
}
```

Output:



A* SEARCH ALGORITHM

NAME : BHUT TUSHAR G
ROLL NO : 20BCP023



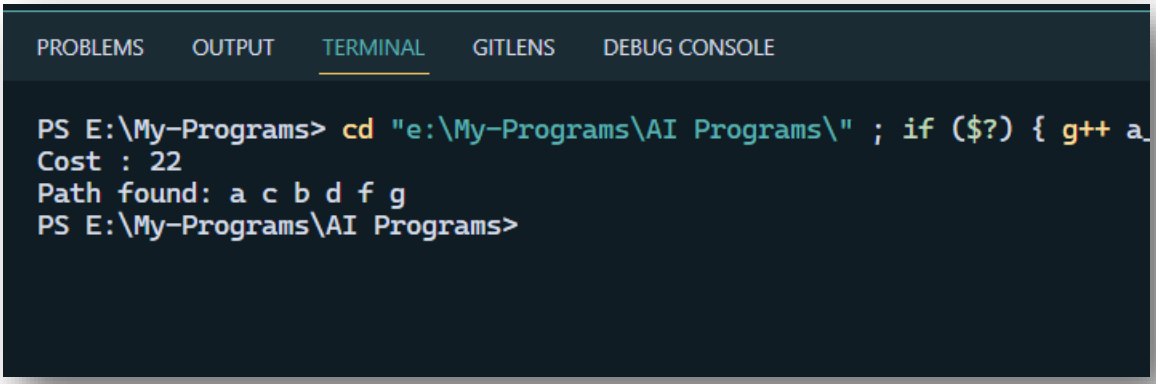
Code :

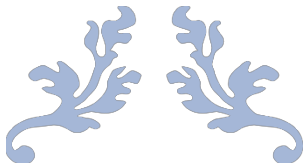
```
/*
   Name : Bhut Tushar
*/

#include <bits/stdc++.h>

#define all(x) (x).begin(), (x).end()
#define SORT(v) sort(all(v))
#define REVSORT(v) sort(all(v),greater<int>())
#define MAX(v) max_element(all(v))
#define MIN(v) min_element(all(v))
#define rep(from,to) for(int i = from;i<=to;i++)
#define rep_back(from,to) for(int i = from;i>=to;i--)
#define take(v) rep(0,v.size())cin>>v[i];
#define FASTIO ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL)
using namespace std;

/* ========== YOUR CODE HERE ========== */

struct ComparePairs
{
   bool operator()(const pair<int, pair<string,string>> &a, const pair<int, pair<string,string>> &b)
   {
      return a.first > b.first; // compares the integers (first element of the pair)
   }
};

struct Node {
```

```
    int g;
    int h;
    string parent;
};

map<string, vector<pair<string, int>>> graph;

vector<pair<string, int>> get_neighbors(string v) {
    if (graph.count(v) > 0)
        return graph[v];
    else
        return {};
}

map<string, int> H_dist = {
        {"A", 16},
        {"B", 4},
        {"C", 16},
        {"D", 2},
        {"E", 1},
        {"F", 4},
        {"G", 0}};

int heuristic(string n) {
    return H_dist[n];
}

void aStarAlgo(string start_node, string stop_node) {
    set<string> open_set;
    set<string> closed_set;
    map<string, Node> nodes_with_values;
    map<string, string> parents;
    open_set.insert(start_node);
    nodes_with_values[start_node].g = 0;
    parents[start_node] = start_node;

    while (open_set.size() > 0) {
        string n;
        int min_score = INT_MAX;

        for (auto v : open_set) {
            int score = nodes_with_values[v].g + heuristic(v);
            if (score < min_score) {
                n = v;
                min_score = score;
            }
        }

        if (n.empty()) {
            cout << "Path does not exist!" << endl;
            return;
        }

        if (n == stop_node) {
```

```
            vector<string> path;
            while (parents[n] != n) {
                path.push_back(n);
                n = parents[n];
            }
            path.push_back(start_node);
            reverse(all(path));
            cout<<"Cost : "<<nodes_with_values[stop_node].g<<"\n";
            cout << "Path found: ";
            for (auto s : path)
                cout << s << " ";
            cout << endl;
            return;
        }
        open_set.erase(n);
        closed_set.insert(n);
        vector<pair<string, int>> neighbors = get_neighbors(n);
        if (neighbors.empty())
            continue;

        for (auto m : neighbors) {
            if (open_set.count(m.first) == 0 && closed_set.count(m.first) == 0) {
                open_set.insert(m.first);
                parents[m.first] = n;
                nodes_with_values[m.first].g = nodes_with_values[n].g + m.second;
            } else {
                if (nodes_with_values[m.first].g > nodes_with_values[n].g + m.second) {
                    nodes_with_values[m.first].g = nodes_with_values[n].g + m.second;
                    parents[m.first] = n;
                    if (closed_set.count(m.first) > 0) {
                        open_set.insert(m.first);
                        closed_set.erase(m.first);
                    }
                }
            }
        }
    }
    cout << "Path does not exist!" << endl;
    return;
}
int main() {
    graph["a"] = {{"b", 10}, {"c", 4}};
    graph["b"] = {{"a", 10}, {"d", 6}, {"c", 4}};
    graph["c"] = {{"a", 4}, {"b", 4}, {"e", 6}};
    graph["d"] = {{"b", 6}, {"e", 6}, {"f", 4}};
    graph["e"] = {{"c", 6}, {"d", 6}, {"g", 14}};
    graph["f"] = {{"d", 4}, {"g", 4}};
    graph["g"] = {{"e", 14}, {"f", 4}};
    aStarAlgo("a", "g");
    return 0;
}
```

Output :


PROBLEMS
OUTPUT
TERMINAL
GITLENS
DEBUG CONSOLE
PS E:\My-Programs> cd "e:\My-Programs\AI Programs\" ; if ($?) { g++ a
Cost : 22
Path found: a c b d f g
PS E:\My-Programs\AI Programs>



8 PUZZLE USING A* ALGORITHM

NAME : BHUT TUSHAR G.
ROLL NO : 20BCP023

Solve the given 8-puzzle problem by using A* algorithm.

Initial State

Final State

Where g(n) = Depth of node and

h(n) = Number of misplaced tiles.

Code:

```
#include <bits/stdc++.h>

#define all(x) (x).begin(), (x).end()

#define FASTIO                      \
    ios_base::sync_with_stdio(false); \
    cin.tie(NULL);                  \
    cout.tie(NULL)
using namespace std;

/* ========== YOUR CODE HERE ========== */

int size;
class Node
{
public:
    vector<vector<int>> board;
    int depth, misplaced;
    Node *parent;
    Node()
    {
        depth = 0;
        misplaced = 0;
        parent = NULL;
    }

    static int heuristic(Node start, Node goal)
    {
        int count = 0;
        for (int i = 0; i < start.board.size(); i++)
        {
            for (int j = 0; j < start.board.size(); j++)
            {
```

```
            if (start.board[i][j] != 0 and start.board[i][j] != goal.board[i][j])
            {
                count++;
            }
        }
    }
    return count;
}
bool operator==(Node a)
{
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            if (this->board[i][j] != a.board[i][j])
                return false;
    return true;
}
void print()
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
            cout << board[i][j] << " ";
        cout << endl;
    }
}
};

bool compare(Node a, Node b)
{
    return a.misplaced < b.misplaced;
}

bool isinset(Node a, vector<Node> b)
{
    for (int i = 0; i < b.size(); i++)
        if (a == b[i])
            return true;
    return false;
}

void addChild(Node current, Node goal, int newi, int newj, int blanki, int blankj, vector<Node> &openset,
vector<Node> &closeset)
{
    Node newNode = current;

    swap(newNode.board[newi][newj], newNode.board[blanki][blankj]);
    if (!isinset(newNode, openset) and !isinset(newNode, closeset))
    {
        newNode.depth = current.depth + 1;
        newNode.misplaced = newNode.depth + Node::heuristic(newNode, goal);
        Node *temp = new Node;
        *temp = current;
        newNode.parent = temp;
        openset.push_back(newNode);
```

```
    }
}

void getChilds(Node current, Node goal, vector<Node> &openset, vector<Node> &closeset)
{
    int blanki, blankj;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (current.board[i][j] == 0)
            {
                blanki = i;
                blankj = j;
                break;
            }
        }
    }
    int i = blanki;
    int j = blankj;

    if (i - 1 >= 0)
    {
        addChild(current, goal, i - 1, j, blanki, blankj, openset, closeset);
    }
    if (i + 1 < size)
    {
        addChild(current, goal, i + 1, j, blanki, blankj, openset, closeset);
    }
    if (j - 1 >= 0)
    {
        addChild(current, goal, i, j - 1, blanki, blankj, openset, closeset);
    }
    if (j + 1 < size)
    {
        addChild(current, goal, i, j + 1, blanki, blankj, openset, closeset);
    }
}

bool reconstruct_path(Node current, vector<Node> &came_from)
{
    // cout << "Generating path ...\n";
    Node *temp = &current;
    while (temp != NULL)
    {
        came_from.push_back(*temp);
        temp = temp->parent;
    }
    return true;
}

vector<Node> path;
bool solve(Node start, Node goal)
{
```

```
    vector<Node> openset;
    vector<Node> closeset;

    start.depth = 0;
    start.misplaced = start.depth + Node ::heuristic(start, goal);
    openset.push_back(start);

    while (!openset.empty())
    {
        sort(all(openset), compare);
        Node current = openset[0];

        if (current == goal)
        {
            return reconstruct_path(current, path);
        }
        openset.erase(openset.begin());
        closeset.push_back(current);

        getChilds(current, goal, openset, closeset);
    }
    return false;
}

int main()
{
    FASTIO;
    int n = 3;
    size = n;
    Node initial_state;
    initial_state.board = {{2, 8, 3}, {1, 6, 4}, {7, 0, 5}};
    Node goal_state;
    goal_state.board = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}};

    if (solve(initial_state, goal_state))
    {
        cout << "Solved\n";
        cout << "\n***********\n";
        for (int i = path.size() - 1; i >= 0; i--)
        {
            path[i].print();
            if (i != 0)
                cout << "\n  |\n\n";
        }
        cout << "\n***********\n";
    }
    else
    {
        cout << "Fail\n";
    }

    return 0;
}
```

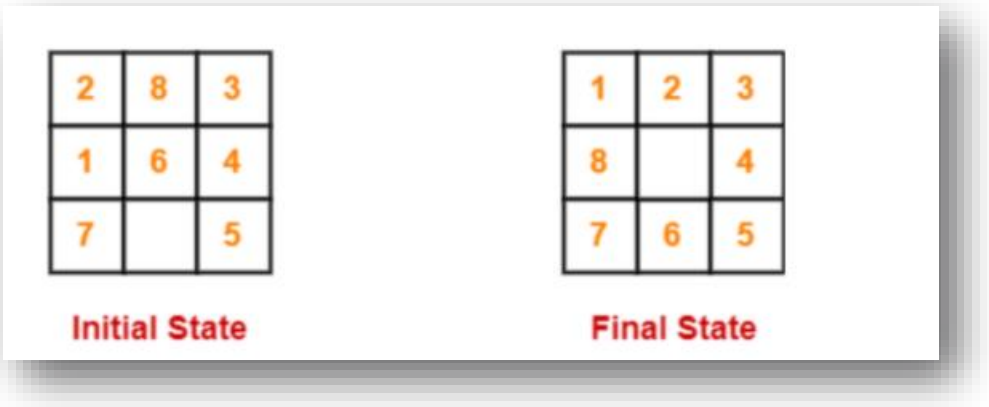
Output :


PROBLEMS
OUTPUT
TERMINAL
GITLENS
COMMENTS
DEBUG CONSOLE
Code - AI Programs
PS E:\My-Programs> cd "e:\My-Programs\AI Programs\" ; if ($?) { g++ eight_puzzle.cpp -o eight_puzzle } ; if ($?) { .\eight_puzzle }
Solved
****
2 8 3
1 6 4
7 0 5
|
2 8 3
1 0 4
7 6 5
|
2 0 3
1 8 4
7 6 5
|
0 2 3
1 8 4
7 6 5
|
1 2 3
0 8 4
7 6 5
|
1 2 3
8 0 4
7 6 5
****
PS E:\My-Programs\AI Programs>

8 PUZZLE USING A* ALGORITHM & HILL CLIMBING

NAME : BHUT TUSHAR G.
ROLL NO : 20BCP023

Solve the given 8-puzzle problem by using A* and hill climbing algorithm and compare the time taken by both the algorithm.



A* Algorithm :

```
#include <bits/stdc++.h>
#include <chrono>
using namespace std::chrono;
#define all(x) (x).begin(), (x).end()

#define FASTIO                          \
    ios_base::sync_with_stdio(false); \
    cin.tie(NULL);                    \
    cout.tie(NULL)
using namespace std;

/* ========== YOUR CODE HERE ========= */

int size;
class Node
{
public:
    vector<vector<int>> board;
    int depth, misplaced;
    Node *parent;
    Node()
    {
        depth = 0;
        misplaced = 0;
        parent = NULL;
    }

    static int heuristic(Node start, Node goal)
    {
        int count = 0;
        for (int i = 0; i < start.board.size(); i++)
        {
            for (int j = 0; j < start.board.size(); j++)
            {
                if (start.board[i][j] != 0 and start.board[i][j] != goal.board[i][j])
                {
```

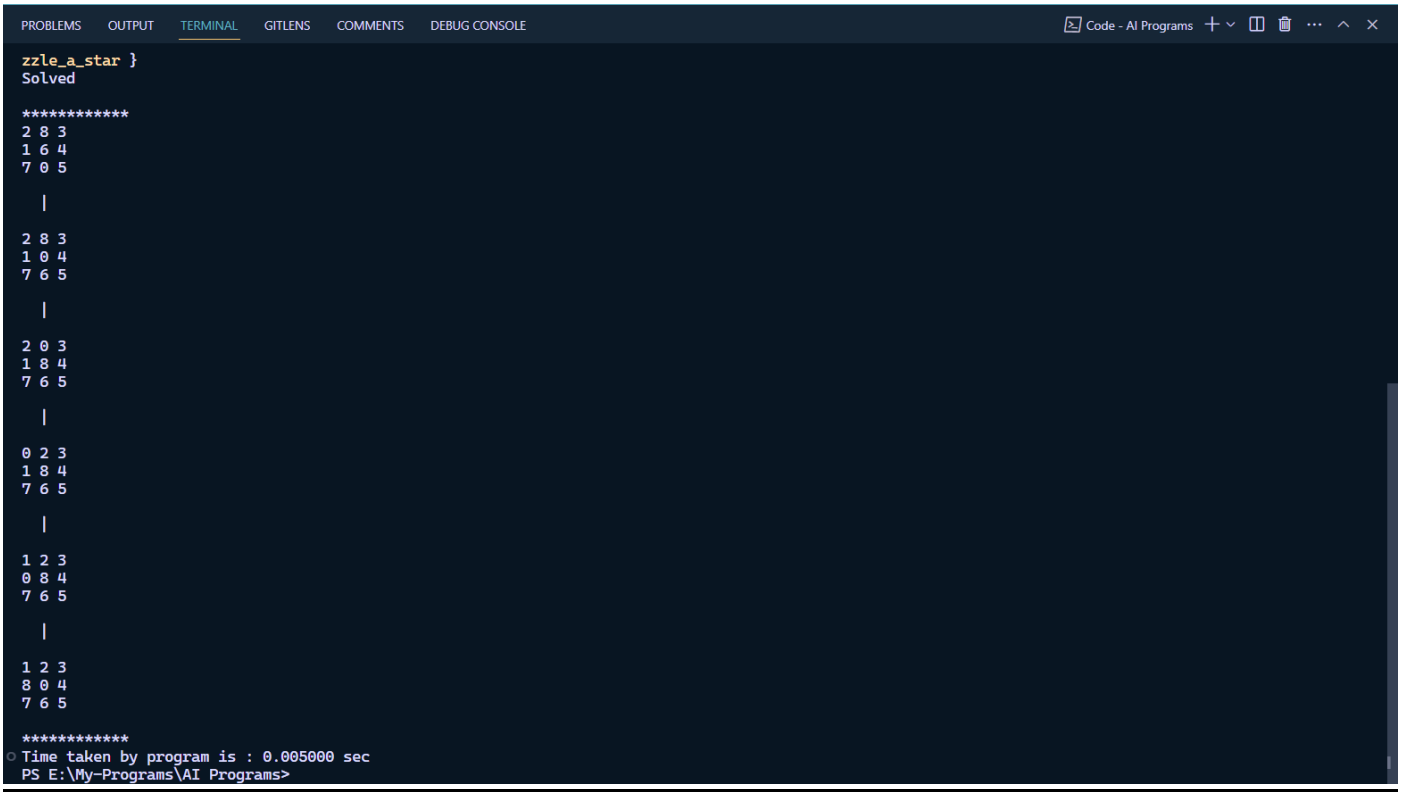
```
                    count++;
                }
            }
        }
        return count;
    }
    bool operator==(Node a)
    {
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
                if (this->board[i][j] != a.board[i][j])
                    return false;
        return true;
    }
    void print()
    {
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
                cout << board[i][j] << " ";
            cout << endl;
        }
    }
};

bool compare(Node a, Node b)
{
    return a.misplaced < b.misplaced;
}

bool isinset(Node a, vector<Node> b)
{
    for (int i = 0; i < b.size(); i++)
        if (a == b[i])
            return true;
    return false;
}

void addChild(Node current, Node goal, int newi, int newj, int blanki, int blankj,
vector<Node> &openset, vector<Node> &closeset)
{
    Node newNode = current;

    swap(newNode.board[newi][newj], newNode.board[blanki][blankj]);
    if (!isinset(newNode, openset) and !isinset(newNode, closeset))
    {
        newNode.depth = current.depth + 1;
        newNode.misplaced = newNode.depth + Node::heuristic(newNode, goal);
        Node *temp = new Node;
        *temp = current;
        newNode.parent = temp;
        openset.push_back(newNode);
    }
}
```

```
void getChilds(Node current, Node goal, vector<Node> &openset, vector<Node>
&closeset)
{
    int blanki, blankj;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (current.board[i][j] == 0)
            {
                blanki = i;
                blankj = j;
                break;
            }
        }
    }
    int i = blanki;
    int j = blankj;

    if (i - 1 >= 0)
    {
        addChild(current, goal, i - 1, j, blanki, blankj, openset, closeset);
    }
    if (i + 1 < size)
    {
        addChild(current, goal, i + 1, j, blanki, blankj, openset, closeset);
    }
    if (j - 1 >= 0)
    {
        addChild(current, goal, i, j - 1, blanki, blankj, openset, closeset);
    }
    if (j + 1 < size)
    {
        addChild(current, goal, i, j + 1, blanki, blankj, openset, closeset);
    }
}

bool reconstruct_path(Node current, vector<Node> &came_from)
{
    // cout << "Generating path ...\n";
    Node *temp = &current;
    while (temp != NULL)
    {
        came_from.push_back(*temp);
        temp = temp->parent;
    }
    return true;
}

vector<Node> path;
bool solve(Node start, Node goal)
{
    vector<Node> openset;
```

```
    vector<Node> closeset;

    start.misplaced = start.depth + Node ::heuristic(start, goal);
    openset.push_back(start);

    while (!openset.empty())
    {
        sort(all(openset), compare);
        Node current = openset[0];

        if (current == goal)
        {
            return reconstruct_path(current, path);
        }
        openset.erase(openset.begin());
        closeset.push_back(current);

        getChilds(current, goal, openset, closeset);
    }
    return false;
}

int main()
{
    FASTIO;

    int n = 3;
    size = n;
    Node initial_state;
    initial_state.board = {{2, 8, 3}, {1, 6, 4}, {7, 0, 5}};
    // initial_state.board = {{1, 2, 3}, {5, 6, 0}, {7, 8, 4}};
    // initial_state.board = {{1, 2, 3}, {4, 7, 6}, {5, 0, 8}};
    // initial_state.board = {{2, 8, 3}, {1, 6, 4}, {7, 0, 5}};
    Node goal_state;
    goal_state.board = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}};
    // goal_state.board = {{1, 2, 3}, {4, 5, 6}, {7, 0, 8}};
        // goal_state.board = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}};

     clock_t start, end;

    /* Recording the starting clock tick.*/
    start = clock();
    if (solve(initial_state, goal_state))
    {
        cout << "Solved\n";
        cout << "\n************\n";
        for (int i = path.size() - 1; i >= 0; i--)
        {
            path[i].print();
            if (i != 0)
                cout << "\n  | \n\n";
        }
        cout << "\n************\n";
    }
```

```
    else
    {
        cout << "Fail\n";
    }
    end = clock();

    // Calculating total time taken by the program.
    double time_taken = double(end - start) / double(CLOCKS_PER_SEC);
    cout << "Time taken by program is : " << fixed
         << time_taken << setprecision(5);
    cout << " sec " << endl;
    return 0;
}
```

Output : Time taken : 0.005000 sec



Hill Climbing:

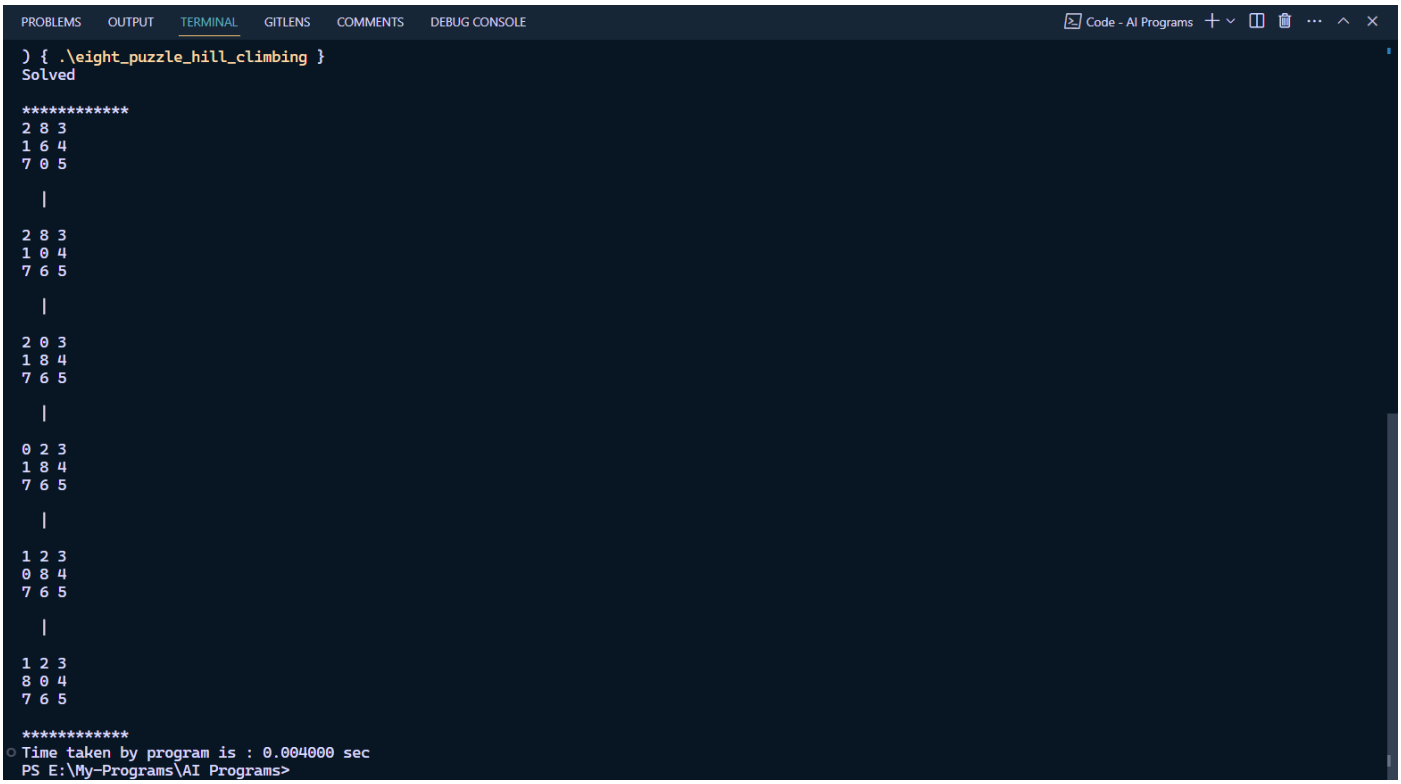
```
#include <bits/stdc++.h>
#include <chrono>
using namespace std::chrono;
#define all(x) (x).begin(), (x).end()

#define FASTIO                          \
    ios_base::sync_with_stdio(false); \
    cin.tie(NULL);                    \
    cout.tie(NULL)
using namespace std;

/* ========== YOUR CODE HERE ========= */

int size;
class Node
{
public:
    vector<vector<int>> board;
    int misplaced;
    Node *parent;
    Node()
    {
        misplaced = 0;
        parent = NULL;
    }

    static int heuristic(Node start, Node goal)
    {
        int count = 0;
        for (int i = 0; i < start.board.size(); i++)
        {
            for (int j = 0; j < start.board.size(); j++)
            {
                if (start.board[i][j] != 0 and start.board[i][j] != goal.board[i][j])
                {
                    count++;
                }
            }
        }
        return count;
    }
    bool operator==(Node a)
    {
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
                if (this->board[i][j] != a.board[i][j])
                    return false;
        return true;
    }
    void print()
    {
        for (int i = 0; i < size; i++)
```

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

bool compare(Node a, Node b)
{
    return a.misplaced < b.misplaced;
}

bool isinset(Node a, vector<Node> b)
{
    for (int i = 0; i < b.size(); i++)
        if (a == b[i])
            return true;
    return false;
}

void addChild(Node current, Node goal, int newi, int newj, int blanki, int blankj,
vector<Node> &openset, vector<Node> &closeset)
{
    Node newNode = current;

    swap(newNode.board[newi][newj], newNode.board[blanki][blankj]);
    if (!isinset(newNode, openset) and !isinset(newNode, closeset))
    {
        newNode.misplaced = Node::heuristic(newNode, goal);
        Node *temp = new Node;
        *temp = current;
        newNode.parent = temp;
        openset.push_back(newNode);
    }
}

void getChilds(Node current, Node goal, vector<Node> &openset, vector<Node>
&closeset)
{
    int blanki, blankj;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (current.board[i][j] == 0)
            {
                blanki = i;
                blankj = j;
                break;
            }
        }
    }
    int i = blanki;
```

```
    int j = blankj;

    if (i - 1 >= 0)
    {
        addChild(current, goal, i - 1, j, blanki, blankj, openset, closeset);
    }
    if (i + 1 < size)
    {
        addChild(current, goal, i + 1, j, blanki, blankj, openset, closeset);
    }
    if (j - 1 >= 0)
    {
        addChild(current, goal, i, j - 1, blanki, blankj, openset, closeset);
    }
    if (j + 1 < size)
    {
        addChild(current, goal, i, j + 1, blanki, blankj, openset, closeset);
    }
}

bool reconstruct_path(Node current, vector<Node> &came_from)
{
    // cout << "Generating path ...\n";
    Node *temp = &current;
    while (temp != NULL)
    {
        came_from.push_back(*temp);
        temp = temp->parent;
    }
    return true;
}

vector<Node> path;
bool solve(Node start, Node goal)
{
    vector<Node> openset;
    vector<Node> closeset;

    start.misplaced = Node ::heuristic(start, goal);
    openset.push_back(start);

    while (!openset.empty())
    {
        sort(all(openset), compare);
        Node current = openset[0];
        openset.clear();
        if (current == goal)
        {
            return reconstruct_path(current, path);
        }
        closeset.push_back(current);

        getChilds(current, goal, openset, closeset);
    }
```

```
    return false;
}

int main()
{
    FASTIO;

    int n = 3;
    size = n;
    Node initial_state;
    // initial_state.board = {{1, 2, 3}, {4, 7, 6}, {5, 0, 8}};
    // initial_state.board = {{1, 2, 3}, {5, 6, 0}, {7, 8, 4}};
    initial_state.board = {{2, 8, 3}, {1, 6, 4}, {7, 0, 5}};
    Node goal_state;
    // goal_state.board = {{1, 2, 3}, {4, 5, 6}, {7, 0, 8}};
    // goal_state.board = {{1, 2, 3}, {4, 5, 6}, {7, 0, 8}};
    goal_state.board = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}};
    clock_t start, end;

    /* Recording the starting clock tick.*/
    start = clock();
    if (solve(initial_state, goal_state))
    {
        cout << "Solved\n";
        cout << "\n************\n";
        for (int i = path.size() - 1; i >= 0; i--)
        {
            path[i].print();
            if (i != 0)
                cout << "\n  | \n\n";
        }
        cout << "\n************\n";
    }
    else
    {
        cout << "Fail\n";
    }
    end = clock();

    // Calculating total time taken by the program.
    double time_taken = double(end - start) / double(CLOCKS_PER_SEC);
    cout << "Time taken by program is : " << fixed
         << time_taken << setprecision(5);
    cout << " sec " << endl;
    return 0;
}
```

Output: Time taken: 0.004000 sec



```
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
import seaborn as sns
import matplotlib.pyplot as plt
from keras.layers import Dense
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data= pd.read_csv('/content/drive/MyDrive/indian_liver_patient.csv')
data
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferas
0	65	Female	0.7	0.1	187	1
1	62	Male	10.9	5.5	699	6
2	62	Male	7.3	4.1	490	6
3	58	Male	1.0	0.4	182	1
4	72	Male	3.9	2.0	195	2
...	...	...	...	...	...	.
578	60	Male	0.5	0.1	500	2
579	40	Male	0.6	0.1	98	3
580	52	Male	0.8	0.2	245	4
581	31	Male	1.3	0.5	184	2
582	38	Male	1.0	0.3	216	2

583 rows × 11 columns

```
data.describe
```

```
<bound method NDFrame.describe of     Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0    65  Female              0.7               0.1                   187
1    62    Male             10.9               5.5                   699
2    62    Male              7.3               4.1                   490
3    58    Male              1.0               0.4                   182
4    72    Male              3.9               2.0                   195
..  ...     ...              ...               ...                   ...
578  60    Male              0.5               0.1                   500
579  40    Male              0.6               0.1                    98
580  52    Male              0.8               0.2                   245
581  31    Male              1.3               0.5                   184
582  38    Male              1.0               0.3                   216

     Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                          16                          18             6.8
1                          64                         100             7.5
2                          60                          68             7.0
3                          14                          20             6.8
4                          27                          59             7.3
..                        ...                         ...             ...
578                        20                          34             5.9
579                        35                          31             6.0
580                        48                          49             6.4
581                        29                          32             6.8
582                        21                          24             7.3

     Albumin  Albumin_and_Globulin_Ratio  Dataset
0        3.3                        0.90        1
1        3.2                        0.74        1
2        3.3                        0.89        1
3        3.4                        1.00        1
4        2.4                        0.40        1
..       ...                         ...      ...
```

```
578    1.6                  0.37      2
579    3.2                  1.10      1
580    3.2                  1.00      1
581    3.4                  1.00      1
582    4.4                  1.50      2

[583 rows x 11 columns]>
```

```
data.isnull().sum()
```

```
Age                         0
Gender                      0
Total_Bilirubin             0
Direct_Bilirubin            0
Alkaline_Phosphotase        0
Alamine_Aminotransferase    0
Aspartate_Aminotransferase  0
Total_Protiens              0
Albumin                     0
Albumin_and_Globulin_Ratio  4
Dataset                     0
dtype: int64
```

```
data.Albumin_and_Globulin_Ratio.fillna(data.Albumin_and_Globulin_Ratio.mean(),inplace=True)
```

```
data.isnull().sum()
```

```
Age                         0
Gender                      0
Total_Bilirubin             0
Direct_Bilirubin            0
Alkaline_Phosphotase        0
Alamine_Aminotransferase    0
Aspartate_Aminotransferase  0
Total_Protiens              0
Albumin                     0
Albumin_and_Globulin_Ratio  0
Dataset                     0
dtype: int64
```

```
data.dtypes
```

```
Age                           int64
Gender                       object
Total_Bilirubin             float64
Direct_Bilirubin            float64
Alkaline_Phosphotase          int64
Alamine_Aminotransferase      int64
Aspartate_Aminotransferase    int64
Total_Protiens              float64
Albumin                     float64
Albumin_and_Globulin_Ratio  float64
Dataset                       int64
dtype: object
```

```
from sklearn.preprocessing import LabelEncoder,StandardScaler
```

```
enc = LabelEncoder()
```

```
data.Gender = enc.fit_transform(data['Gender'])
```

```
data.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminot
0	65	0	0.7	0.1	187	16	
1	62	1	10.9	5.5	699	64	
2	62	1	7.3	4.1	490	60	
3	58	1	1.0	0.4	182	14	
4	72	1	3.9	2.0	195	27	

```
sc = StandardScaler()
```

```
for feature in data[['Age','Total_Bilirubin','Direct_Bilirubin','Alkaline_Phosphotase','Alamine_Aminotransferase','Aspartate_Aminotransferase','Total_Protiens','Albumin']]:
  data[feature] = sc.fit_transform(data[feature].values.reshape(-1, 1))
```

```
data.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_A
0	1.252098	0	-0.418878	-0.493964	-0.426715	-0.354665	
1	1.066637	1	1.225171	1.430423	1.682629	-0.091599	
2	1.066637	1	0.644919	0.931508	0.821588	-0.113522	
3	0.819356	1	-0.370523	-0.387054	-0.447314	-0.365626	
4	1.684839	1	0.096902	0.183135	-0.393756	-0.294379	

```
data.Dataset.value_counts()
```

```
1   416
2   167
Name: Dataset, dtype: int64
```

```
X = data.drop(['Dataset','Gender'],axis=1)
X.head()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Asp
0	1.252098	-0.418878	-0.493964	-0.426715	-0.354665	
1	1.066637	1.225171	1.430423	1.682629	-0.091599	
2	1.066637	0.644919	0.931508	0.821588	-0.113522	
3	0.819356	-0.370523	-0.387054	-0.447314	-0.365626	
4	1.684839	0.096902	0.183135	-0.393756	-0.294379	

```
Y = data['Dataset']
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
```

```
xtrain,xtest,ytrain,ytest = train_test_split(X,Y,random_state=0,stratify=Y)
```

```
class Perceptron:
 def __init__(self):
  self.w = None
  self.b = None

 def model(self,x):
  return 1 if np.dot(self.w,x)>=self.b  else 0

 def predict(self,X):
  Y = []
  for x in X:
   result = self.model(x)
   Y.append(result)
  return np.array(Y)

 def fit(self,X,Y,epochs=1000,lr=0.01):
  self.w = np.ones(X.shape[1])
  self.b = 0
  accuracy={}
  max_accuracy = 0
  for i in range(epochs):
   for x,y in zip(X,Y):
```

```
      y_pred = self.model(x)
      if y == 1 and y_pred == 0:
        self.w = self.w+x*lr
        self.b = self.b-1*lr
      elif y == 0 and y_pred == 1:
        self.w = self.w-x*lr
        self.b = self.b+1*lr
    accuracy[i] = accuracy_score(self.predict(X),Y)
    if accuracy[i] > max_accuracy:
      max_accuracy = accuracy[i]
      chkptw = self.w
      chkptb = self.b
  self.w = chkptw
  self.b = chkptb
  print('Max accuracy is ',max_accuracy)
```

```
model = Perceptron()
```

```
xtrain = xtrain.values
ytrain = ytrain.values
model.fit(xtrain,ytrain)
```

```
Max accuracy is  0.7139588100686499
```

```
```

VEHICLE DETECTION

NAME : BHUT TUSHAR G.
ROLL NO : 20BCP023

```
from google.colab import files
files.upload()
```

```
Choose Files kaggle.json
- kaggle.json(application/json) - 63 bytes, last modified: 4/4/2023 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"mrbt007","key":"e86ebc1b5fbde2205e123242f53d7a15"}'}
```

```
!pip install -q kaggle
```

```
!mkdir -p ~/.kaggle
```

```
!cp kaggle.json ~/.kaggle/
```

```
!chmod 600 ~/.kaggle/kaggle.json
```

```
!mkdir vehicle
```

```
%cd vehicle
```

```
/content/vehicle
```

```
!kaggle datasets download -d brsdincer/vehicle-detection-image-set
```

```
Downloading vehicle-detection-image-set.zip to /content/vehicle
 98% 117M/119M [00:06<00:00, 24.9MB/s]
100% 119M/119M [00:06<00:00, 18.2MB/s]
```

```
!unzip vehicle-detection-image-set.zip
```

```
inflating: data/vehicles/right (87).png
inflating: data/vehicles/right (88).png
inflating: data/vehicles/right (89).png
inflating: data/vehicles/right (9).png
inflating: data/vehicles/right (90).png
inflating: data/vehicles/right (91).png
inflating: data/vehicles/right (92).png
inflating: data/vehicles/right (93).png
inflating: data/vehicles/right (94).png
inflating: data/vehicles/right (95).png
inflating: data/vehicles/right (96).png
inflating: data/vehicles/right (97).png
inflating: data/vehicles/right (98).png
inflating: data/vehicles/right (99).png
```

```
!rm vehicle-detection-image-set.zip
!rm sample_submission.csv
```

```
rm: cannot remove 'sample_submission.csv': No such file or directory
```

```
%cd ..
```

```
/content
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
from keras.utils import img_to_array,array_to_img,to_categorical
from keras.models import Sequential
from keras.layers import Conv2D,MaxPool2D,Dense,Flatten
from sklearn.model_selection import train_test_split
```

```
path = '/content/vehicle/data'
root_dir = os.listdir(path)
root_dir
```

```
['non-vehicles', 'vehicles']
```

```
images = [ ]
labels = [ ]
for idx, category in enumerate(root_dir):
  category_path = os.path.join(path, category)
  for img_name in os.listdir(category_path):
    img_path = os.path.join(category_path, img_name)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (64, 64))
    img = img / 255.0
    images.append(img)
    labels.append(idx)

images = np.array(images)
labels = np.array(labels)
```

```
# 0 - vehicle
# 1 - non-vehicle
np.unique(labels,return_counts=True)
```

```
(array([0, 1]), array([8968, 8792]))
```

```
len(images)
```

```
17760
```

```
images[0]
```

```
array([[[0.21176471, 0.20784314, 0.21960784],
        [0.2       , 0.19607843, 0.20784314],
        [0.18039216, 0.17647059, 0.18431373],
        ...,
        [0.59215686, 0.65490196, 0.71764706],
        [0.59215686, 0.65490196, 0.71372549],
        [0.58823529, 0.64705882, 0.70588235]],
```

```
[[0.09411765, 0.10588235, 0.11764706],
 [0.11372549, 0.11764706, 0.12941176],
 [0.09803922, 0.10196078, 0.11372549],
 ...,
 [0.5372549 , 0.58823529, 0.61960784],
 [0.43921569, 0.48627451, 0.52156863],
 [0.34117647, 0.38431373, 0.41960784]],

[[0.12156863, 0.1372549 , 0.15294118],
 [0.15686275, 0.16862745, 0.18431373],
 [0.18431373, 0.19607843, 0.20784314],
 ...,
 [0.18431373, 0.23137255, 0.25490196],
 [0.15686275, 0.20392157, 0.22745098],
 [0.1372549 , 0.18039216, 0.20392157]],

...,

[[0.10196078, 0.04313725, 0.02352941],
 [0.11372549, 0.05490196, 0.03529412],
 [0.11764706, 0.0627451 , 0.04313725],
 ...,
 [0.54509804, 0.56078431, 0.59607843],
 [0.56470588, 0.58039216, 0.61960784],
 [0.57254902, 0.6       , 0.64705882]],

[[0.10980392, 0.05098039, 0.03137255],
 [0.10980392, 0.05490196, 0.03137255],
 [0.10980392, 0.05490196, 0.03137255],
 ...,
 [0.55686275, 0.58039216, 0.61960784],
 [0.56862745, 0.59607843, 0.63921569],
 [0.56470588, 0.6       , 0.65098039]],

[[0.11764706, 0.0627451 , 0.03529412],
 [0.10196078, 0.04705882, 0.02352941],
 [0.09411765, 0.03529412, 0.01176471],
 ...,
 [0.56078431, 0.58431373, 0.61960784],
 [0.56470588, 0.58823529, 0.63529412],
 [0.55686275, 0.59215686, 0.64705882]]])
```

```
from·sklearn.model_selection·import·train_test_split

xtrain,xtest,ytrain,ytest·=·train_test_split(images,labels,test_size=0.2,random_state=0)
```

```
print("xtrain shape:", xtrain.shape)
print("ytrain shape:", ytrain.shape)
print("xtest shape:", xtest.shape)
print("ytest shape:", ytest.shape)
```

```
xtrain shape: (13320, 64, 64, 3)
ytrain shape: (13320,)
xtest shape: (4440, 64, 64, 3)
ytest shape: (4440,)
```

```
from keras.layers import Dense,Conv2D,Flatten,MaxPool2D,MaxPooling2D,Dropout
from keras.optimizers import Adam
```

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer='l2'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer='l2'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer='l2'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
conv2d (Conv2D)           (None, 62, 62, 32)       896

max_pooling2d (MaxPooling2D  (None, 31, 31, 32)      0
)

conv2d_1 (Conv2D)         (None, 29, 29, 64)       18496

max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)      0
2D)

conv2d_2 (Conv2D)         (None, 12, 12, 64)       36928

max_pooling2d_2 (MaxPooling  (None, 6, 6, 64)        0
2D)

conv2d_3 (Conv2D)         (None, 4, 4, 64)         36928

max_pooling2d_3 (MaxPooling  (None, 2, 2, 64)        0
2D)

flatten (Flatten)         (None, 256)              0

dense (Dense)             (None, 128)              32896

dropout (Dropout)         (None, 128)              0

dense_1 (Dense)           (None, 1)                129

=================================================================
Total params: 126,273
Trainable params: 126,273
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
hist = model.fit(xtrain,ytrain,epochs=20,batch_size=32,validation_split=0.2)
```

```
Epoch 1/20
333/333 [==============================] - 15s 8ms/step - loss: 0.6393 - accuracy: 0.8410 - val_loss: 0.3166 - val_accuracy: 0.9298
Epoch 2/20
333/333 [==============================] - 3s 8ms/step - loss: 0.2646 - accuracy: 0.9483 - val_loss: 0.2545 - val_accuracy: 0.9392
Epoch 3/20
333/333 [==============================] - 2s 6ms/step - loss: 0.1740 - accuracy: 0.9716 - val_loss: 0.2248 - val_accuracy: 0.9538
Epoch 4/20
333/333 [==============================] - 2s 6ms/step - loss: 0.1477 - accuracy: 0.9766 - val_loss: 0.1085 - val_accuracy: 0.9869
Epoch 5/20
333/333 [==============================] - 2s 7ms/step - loss: 0.1191 - accuracy: 0.9809 - val_loss: 0.1201 - val_accuracy: 0.9809
Epoch 6/20
333/333 [==============================] - 2s 6ms/step - loss: 0.1093 - accuracy: 0.9806 - val_loss: 0.0949 - val_accuracy: 0.9846
Epoch 7/20
333/333 [==============================] - 2s 7ms/step - loss: 0.1061 - accuracy: 0.9800 - val_loss: 0.1021 - val_accuracy: 0.9805
Epoch 8/20
333/333 [==============================] - 3s 8ms/step - loss: 0.0993 - accuracy: 0.9813 - val_loss: 0.0802 - val_accuracy: 0.9917
Epoch 9/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0905 - accuracy: 0.9845 - val_loss: 0.1415 - val_accuracy: 0.9621
Epoch 10/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0942 - accuracy: 0.9819 - val_loss: 0.1203 - val_accuracy: 0.9726
Epoch 11/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0901 - accuracy: 0.9836 - val_loss: 0.0943 - val_accuracy: 0.9786
Epoch 12/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0844 - accuracy: 0.9840 - val_loss: 0.0939 - val_accuracy: 0.9797
Epoch 13/20
333/333 [==============================] - 3s 8ms/step - loss: 0.0729 - accuracy: 0.9871 - val_loss: 0.0627 - val_accuracy: 0.9917
Epoch 14/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0813 - accuracy: 0.9851 - val_loss: 0.0621 - val_accuracy: 0.9936
Epoch 15/20
333/333 [==============================] - 2s 7ms/step - loss: 0.0687 - accuracy: 0.9879 - val_loss: 0.0659 - val_accuracy: 0.9910
Epoch 16/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0754 - accuracy: 0.9864 - val_loss: 0.0672 - val_accuracy: 0.9902
Epoch 17/20
333/333 [==============================] - 2s 6ms/step - loss: 0.0707 - accuracy: 0.9876 - val_loss: 0.0760 - val_accuracy: 0.9820
Epoch 18/20
333/333 [==============================] - 2s 7ms/step - loss: 0.0682 - accuracy: 0.9878 - val_loss: 0.0622 - val_accuracy: 0.9906
Epoch 19/20
333/333 [==============================] - 4s 11ms/step - loss: 0.0634 - accuracy: 0.9892 - val_loss: 0.0775 - val_accuracy: 0.9835
Epoch 20/20
333/333 [==============================] - 3s 10ms/step - loss: 0.0691 - accuracy: 0.9886 - val_loss: 0.0891 - val_accuracy: 0.9820
```

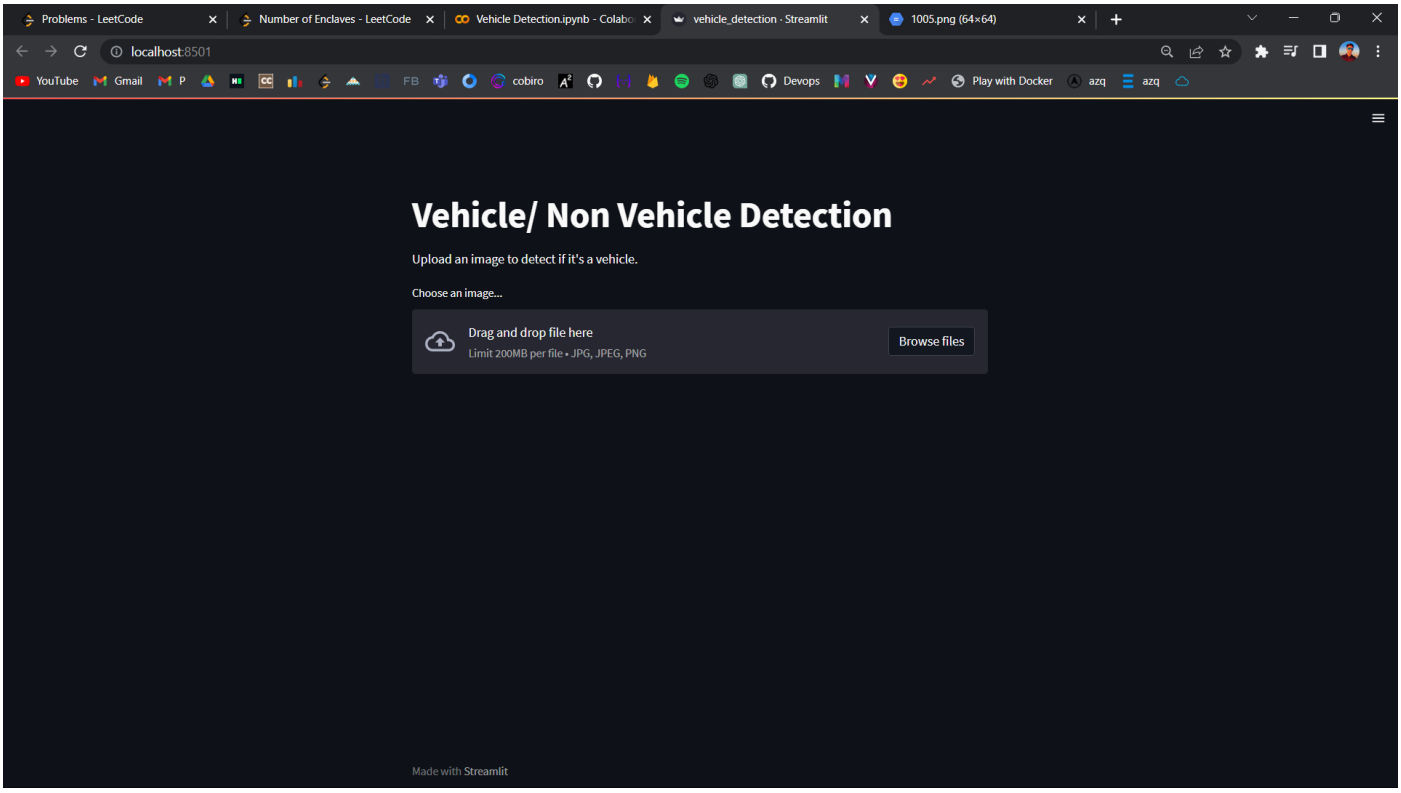
```
model.evaluate(xtest,ytest)
```

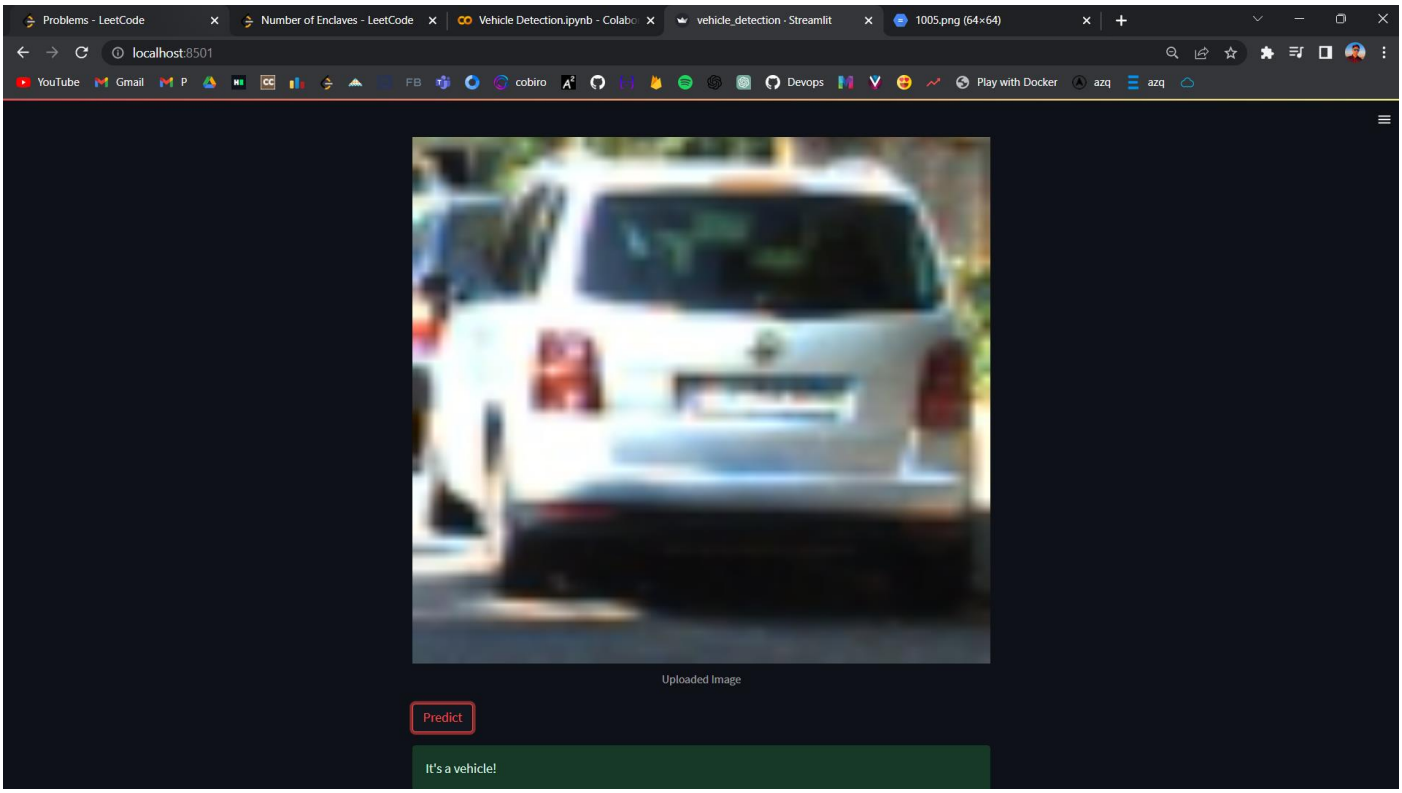
```
139/139 [==============================] - 1s 5ms/step - loss: 0.0795 - accuracy: 0.9838
[0.07951168715953827, 0.9837837815284729]
```

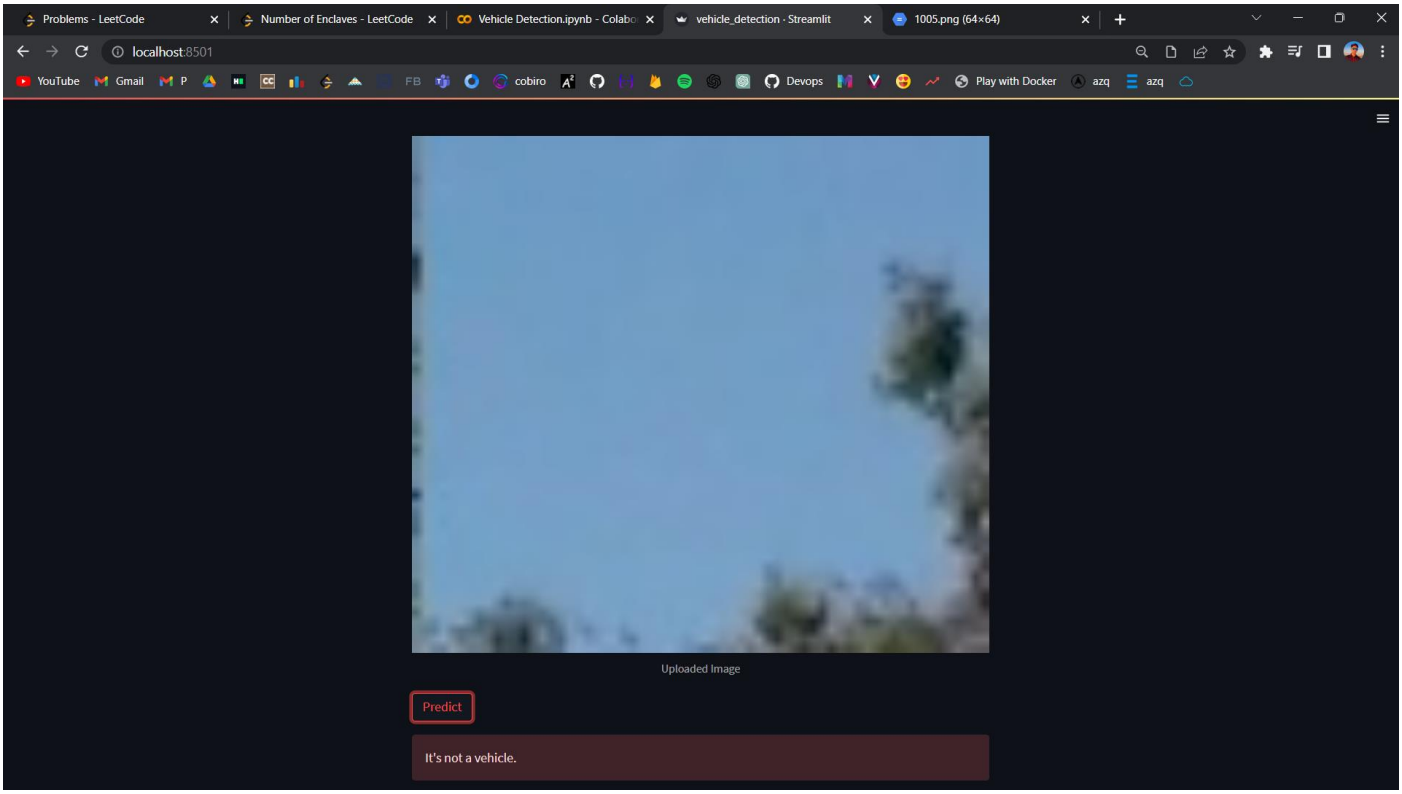
```
model.save('vehicle_detection.h5')
```

```

```

```python
import streamlit as st
import cv2
import numpy as np
import tensorflow as tf
model = tf.keras.models.load_model('vehicle_detection.h5')
st.title("Vehicle/ Non Vehicle Detection")
st.write("Upload an image to detect if it's a vehicle.")
uploaded_file = st.file_uploader(
    "Choose an image...", type=["jpg", "jpeg", "png"])


def predict_vehicle(img):
    img = cv2.resize(img, (64, 64))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    pred = model.predict(img)
    return pred[0][0]


if uploaded_file is not None:
    img = cv2.imdecode(np.frombuffer(uploaded_file.read(), np.uint8), -1)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    st.image(img_rgb, caption="Uploaded Image", use_column_width=True)
    if st.button("Predict"):
        result = predict_vehicle(img_rgb)
        if result > 0.5:
            st.success("It's a vehicle!")
        else:
            st.error("It's not a vehicle.")
```




Problems - LeetCode
Number of Enclaves - LeetCode
Vehicle Detection.ipynb - Colab
vehicle_detection - Streamlit
1005.png (64×64)
localhost:8501
YouTube
Gmail
Devops
Play with Docker
azq
Uploaded Image
Predict
It's a vehicle!


Problems - LeetCode
Number of Enclaves - LeetCode
Vehicle Detection.ipynb - Colab
vehicle_detection - Streamlit
1005.png (64×64)
localhost:8501
YouTube
Gmail
Devops
Play with Docker
azq
Uploaded Image
Predict
It's not a vehicle.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.preprocessing.sequence import TimeseriesGenerator
```

```
data = pd.read_csv("/content/drive/MyDrive/Weather_Bhopal (2).csv")
data
```

	DATE	MAX_TEMP	MIN_TEMP	RAINFALL_24_HRS
0	1/1/2020	18.5	9.7	0.0
1	1/2/2020	20.3	12.0	0.0
2	1/3/2020	20.7	14.0	0.0
3	1/4/2020	20.5	11.0	0.0
4	1/5/2020	17.0	7.2	0.0
...	...	...	...	...
451	3/27/2021	33.7	17.0	0.0
452	3/28/2021	36.8	18.4	0.0
453	3/29/2021	38.9	19.2	0.0
454	3/30/2021	41.0	20.0	0.0
455	3/31/2021	40.2	21.0	0.0

456 rows × 4 columns

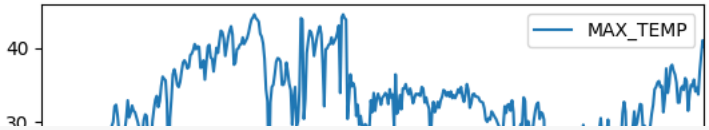
```
data['DATE'] = pd.to_datetime(data['DATE'],infer_datetime_format=True)
```

```
data.describe()
```

	MAX_TEMP	MIN_TEMP	RAINFALL_24_HRS
count	456.000000	456.000000	456.00000
mean	32.059430	18.298465	2.05693
std	6.042076	6.088542	9.35162
min	17.000000	4.600000	0.00000
25%	28.400000	13.400000	0.00000
50%	32.500000	18.600000	0.00000
75%	35.500000	24.000000	0.00000
max	44.500000	29.900000	126.20000

```
data.set_index('DATE')[['MAX_TEMP','MIN_TEMP']].plot(subplots=True)
```

```
array([<Axes: xlabel='DATE'>, <Axes: xlabel='DATE'>], dtype=object)
```




```
input = data[['MAX_TEMP','MIN_TEMP','RAINFALL_24_HRS']]
input.head()
```

	MAX_TEMP	MIN_TEMP	RAINFALL_24_HRS
0	18.5	9.7	0.0
1	20.3	12.0	0.0
2	20.7	14.0	0.0
3	20.5	11.0	0.0
4	17.0	7.2	0.0

```
sca=MinMaxScaler()
data_sca = sca.fit_transform(input)
```

```
x = data_sca
y = data_sca[:,-1]
```

```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,shuffle=False)
```

```
print(xtrain.shape)
print(ytrain.shape)
```

```
(342, 3)
(342,)
```

```
win_len=50
num_feature = 3
train_generator = TimeseriesGenerator(xtrain, ytrain, length=win_len, sampling_rate=1, batch_size=32)
test_generator = TimeseriesGenerator(xtest, ytest, length=win_len, sampling_rate=1, batch_size=32)
```

```
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
import tensorflow as tf
```

```
model = Sequential()
model.add(LSTM(32,input_shape=(win_len,num_feature),activation='tanh',return_sequences=True))
model.add(LSTM(32,activation='tanh',return_sequences=False))
model.add(Dense(1))
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 32)            4608

 lstm_1 (LSTM)               (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 12,961
Trainable params: 12,961
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.callbacks import EarlyStopping
```

```
early_stopping = EarlyStopping(monitor='val_loss',mode='min',patience=3)
```

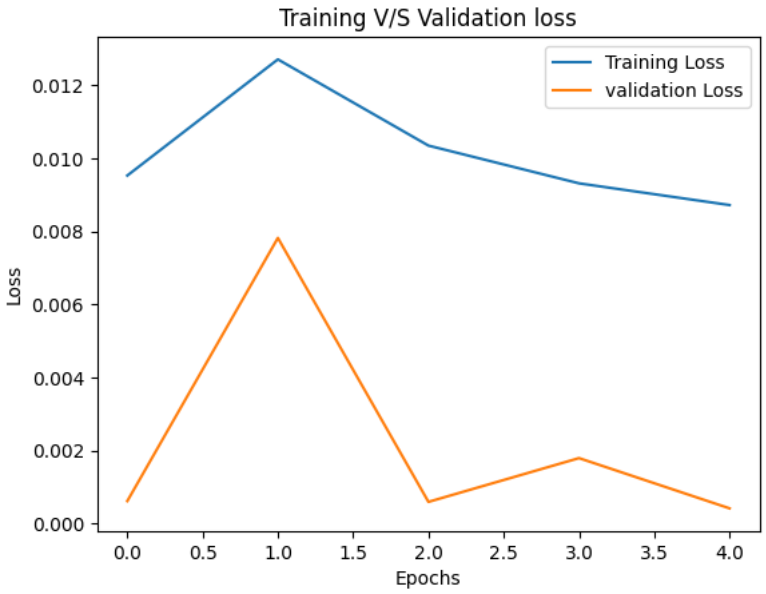
```
model.compile(loss=tf.keras.losses.MeanSquaredError(),optimizer='adam',metrics=[tf.keras.metrics.MeanAbsoluteError()])
```

```
hist = model.fit_generator(train_generator,epochs=5,validation_data=test_generator,callbacks=[early_stopping],shuffle=False)
```

```
Epoch 1/5
<ipython-input-36-cd028c8f5b2c>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generat
  hist = model.fit_generator(train_generator,epochs=5,validation_data=test_generator,callbacks=[early_stopping],shuffle=False)
10/10 [==============================] - 10s 94ms/step - loss: 0.0095 - mean_absolute_error: 0.0622 - val_loss: 6.1389e-04 - val_mean_absolute_error: 0
Epoch 2/5
10/10 [==============================] - 0s 13ms/step - loss: 0.0127 - mean_absolute_error: 0.0666 - val_loss: 0.0078 - val_mean_absolute_error: 0.0863
Epoch 3/5
10/10 [==============================] - 0s 12ms/step - loss: 0.0103 - mean_absolute_error: 0.0576 - val_loss: 5.9175e-04 - val_mean_absolute_error: 0.0
Epoch 4/5
10/10 [==============================] - 0s 12ms/step - loss: 0.0093 - mean_absolute_error: 0.0391 - val_loss: 0.0018 - val_mean_absolute_error: 0.0410
Epoch 5/5
10/10 [==============================] - 0s 12ms/step - loss: 0.0087 - mean_absolute_error: 0.0533 - val_loss: 4.1424e-04 - val_mean_absolute_error: 0.0
```

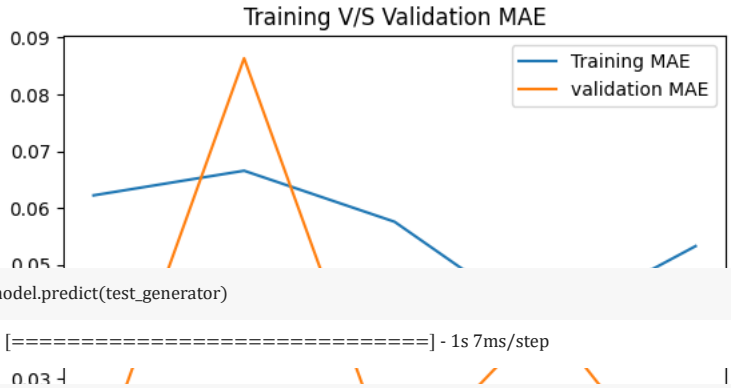
```
import matplotlib.pyplot as plt
plt.plot(hist.history['loss'], label='Training Loss')
plt.plot(hist.history['val_loss'], label='validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title("Training V/S Validation loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f30e02261a0>
```




```
plt.plot(hist.history['mean_absolute_error'], label='Training MAE')
plt.plot(hist.history['val_mean_absolute_error'], label='validation MAE')
plt.xlabel('Epoches')
plt.ylabel('MAE')
plt.title("Training V/S Validation MAE")
plt.legend()
```

<matplotlib.legend.Legend at 0x7f30e02264d0>




```
ypred = model.predict(test_generator)
```

```
2/2 [==============================] - 1s 7ms/step
```

```
print(ypred.shape)
print(xtest.shape)
```

```
(64, 1)
(114, 3)
```

```
df_pred = pd.concat([pd.DataFrame(ypred),pd.DataFrame(xtest[2:,1:][win_len:])],axis=1)
```

```
df_pred
```

	0	0	1
0	-0.004757	0.106719	0.0
1	-0.006371	0.023715	0.0
2	-0.007640	0.071146	0.0
3	-0.008566	0.110672	0.0
4	-0.009457	0.189723	0.0
...	...	...	...
59	0.002770	0.577075	0.0
60	0.002355	0.608696	0.0
61	0.002404	0.648221	0.0
62	0.002834	NaN	NaN
63	0.003588	NaN	NaN

64 rows × 3 columns

```
df_final = sca.inverse_transform(df_pred)
```

```
df_main = input[ypred.shape[0]*-1:]
df_main.shape
```

```
(64, 3)
```

```
df_main['Weather_Pred'] = df_final[:,-1]
df_main
```

<ipython-input-50-3e9813e1162d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
df_main['Weather_Pred'] = df_final[:,-1]

	MAX_TEMP	MIN_TEMP	RAINFALL_24_HRS	Weather_Pred
392	21.5	5.8	0.0	0.0
393	21.9	8.2	0.0	0.0
394	22.4	7.3	0.0	0.0

```
df_main.reset_index(inplace=True)
```

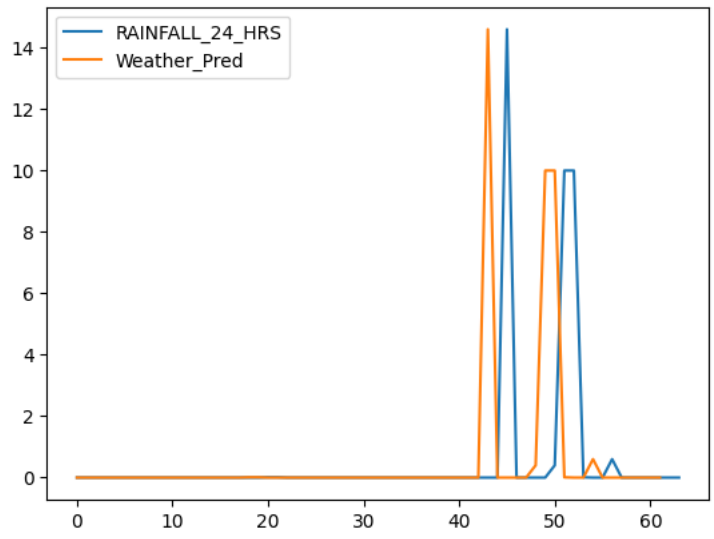
```
df_main
```

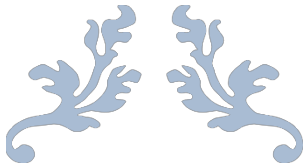
	index	MAX_TEMP	MIN_TEMP	RAINFALL_24_HRS	Weather_Pred
0	392	21.5	5.8	0.0	0.0
1	393	21.9	8.2	0.0	0.0
2	394	22.4	7.3	0.0	0.0
3	395	21.7	5.2	0.0	0.0
4	396	23.7	6.4	0.0	0.0
...	...	...	...	...	...
59	451	33.7	17.0	0.0	0.0
60	452	36.8	18.4	0.0	0.0
61	453	38.9	19.2	0.0	0.0
62	454	41.0	20.0	0.0	NaN
63	455	40.2	21.0	0.0	NaN

64 rows × 5 columns

```
df_main[['RAINFALL_24_HRS','Weather_Pred']].plot()
```

<Axes: >





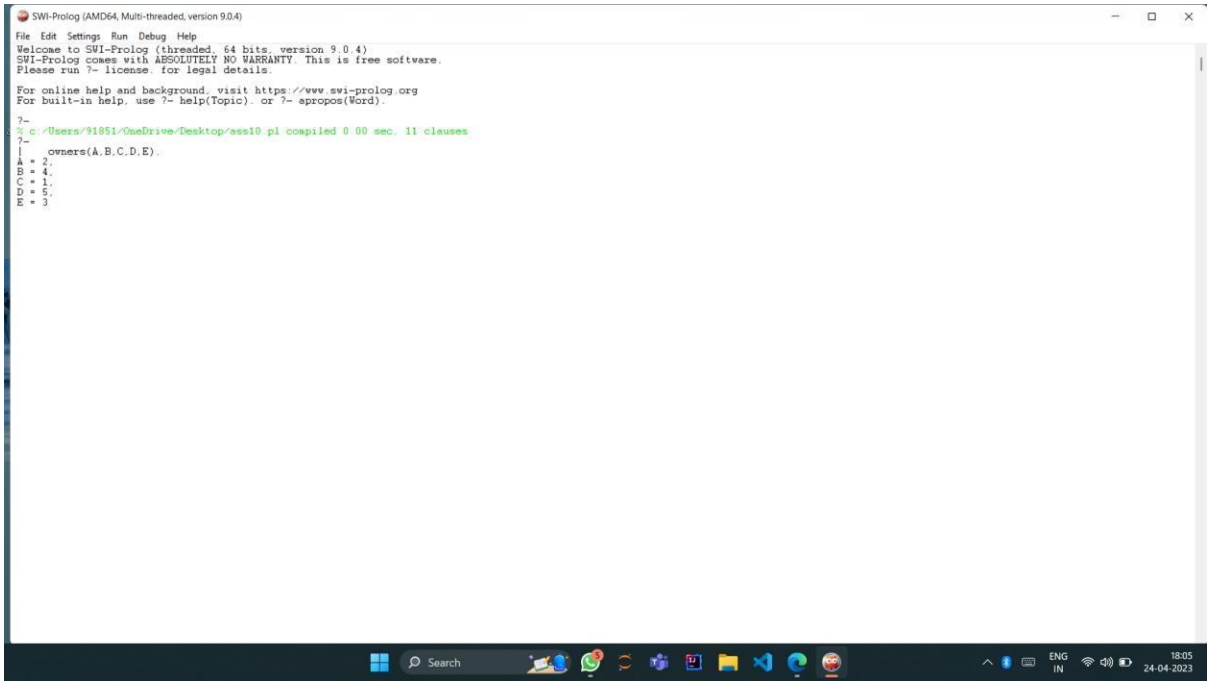
BOX SOLVER PROBLEM

NAME: BHUT TUSHAR G.
ROLL NO: 20BCP023

```
getbox(1). getbox(2). getbox(3). getbox(4). getbox(5).

box(1,black,3).
box(2,black,1).
box(3,white,1).
box(4,black,2).
box(5,white,3). owners(A,B,C,D,E):-
getbox(A),getbox(B),getbox(C),getbox(D),getbox(E),
A\=B,A\=C,A\=D,A\=E,B\=C,B\=D,B\=E,C\=D,C\=E,D\=E,
box(A,Cl1,_),box(B,Cl1,_),
box(D,Cl2,_),box(E,Cl2,_),
box(C,_,S1),box(D,_,S1),
box(E,_,S2),box(B,_,S3), S2<S3.
```

getbox(1). getbox(2). getbox(3). getbox(4). getbox(5).

box(1,black,3).
box(2,black,1).
box(3,white,1).
box(4,black,2).
box(5,white,3). owners(A,B,C,D,E):-
getbox(A),getbox(B),getbox(C),getbox(D),getbox(E),
A\=B,A\=C,A\=D,A\=E,B\=C,B\=D,B\=E,C\=D,C\=E,D\=E,
box(A,Cl1,_),box(B,Cl1,_),
box(D,Cl2,_),box(E,Cl2,_),
box(C,_,S1),box(D,_,S1),
box(E,_,S2),box(B,_,S3), S2<S3.



Name – Bhut Tushar

Roll No – 20BCP023

Assignment- 11

WAP to find the length of the list using Prolog.

```
len([],0).
len([_|T],N):- len(T,N1), N is N1+1.
```