DIGITAL Assignment 2

Name : Lavanya Kushmakar

RegiStRatioN NUmbeR : 22BCE0038

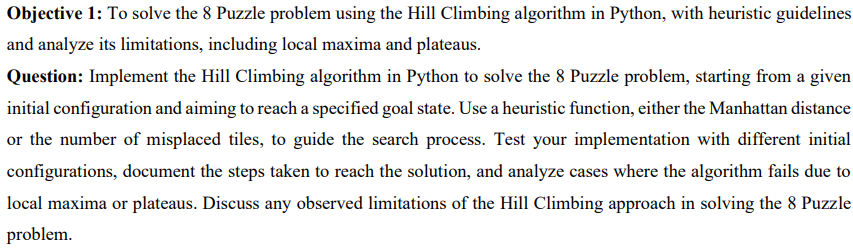
SemeSteR : FALL SEMESTER 2024-25

CoURSe Code : BCSE306L

CoURSe titLe : ARTIFICIAL INTELLIGENCE

FaCULtY Name : SARAVANAGURU RA.K

SLot : E1+TE1

****

**Code in python:**

import copy

import random

from typing import List, Tuple, Optional

class EightPuzzle:

def \_\_init\_\_(self, initial\_state: List[List[int]]):

self.current\_state = initial\_state

self.goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

def find\_empty\_tile(self) -> Tuple[int, int]:

for i in range(3):

for j in range(3):

if self.current\_state[i][j] == 0:

return i, j

return -1, -1

def get\_possible\_moves(self, i: int, j: int) -> List[Tuple[int, int]]:

moves = []

for di, dj in [(0, 1), (1, 0), (0, -1), (-1, 0)]:

new\_i, new\_j = i + di, j + dj

if 0 <= new\_i < 3 and 0 <= new\_j < 3:

moves.append((new\_i, new\_j))

return moves

def make\_move(self, from\_pos: Tuple[int, int], to\_pos: Tuple[int, int]) -> None:

i1, j1 = from\_pos

i2, j2 = to\_pos

self.current\_state[i1][j1], self.current\_state[i2][j2] = \

self.current\_state[i2][j2], self.current\_state[i1][j1]

def manhattan\_distance(self) -> int:

distance = 0

for i in range(3):

for j in range(3):

value = self.current\_state[i][j]

if value != 0:

goal\_i, goal\_j = (value-1) // 3, (value-1) % 3

distance += abs(i - goal\_i) + abs(j - goal\_j)

return distance

def misplaced\_tiles(self) -> int:

count = 0

for i in range(3):

for j in range(3):

if self.current\_state[i][j] != 0 and \

self.current\_state[i][j] != self.goal\_state[i][j]:

count += 1

return count

def hill\_climbing(self, max\_iterations: int = 1000,

heuristic: str = 'manhattan') -> Tuple[bool, int]:

steps = 0

while steps < max\_iterations:

current\_score = self.manhattan\_distance() if heuristic == 'manhattan' \

else self.misplaced\_tiles()

if current\_score == 0:

return True, steps

empty\_i, empty\_j = self.find\_empty\_tile()

possible\_moves = self.get\_possible\_moves(empty\_i, empty\_j)

best\_score = current\_score

best\_move = None

for move in possible\_moves:

self.make\_move((empty\_i, empty\_j), move)

new\_score = self.manhattan\_distance() if heuristic == 'manhattan' \

else self.misplaced\_tiles()

self.make\_move(move, (empty\_i, empty\_j))

if new\_score < best\_score:

best\_score = new\_score

best\_move = move

if best\_move is None:

return False, steps

self.make\_move((empty\_i, empty\_j), best\_move)

steps += 1

return False, steps

def print\_board(state: List[List[int]]) -> None:

for row in state:

print(row)

print()

def get\_user\_input():

print("Enter the initial state of the 8-puzzle (use 0 for empty tile)")

print("Enter numbers row by row (space-separated):")

initial\_state = []

for i in range(3):

row = list(map(int, input(f"Enter row {i+1}: ").strip().split()))

initial\_state.append(row)

return initial\_state

def main():

print("8-Puzzle Solver using Hill Climbing")

initial\_state = get\_user\_input()

print("\nSelect heuristic:")

print("1. Manhattan Distance")

print("2. Misplaced Tiles")

choice = input("Enter choice (1 or 2): ")

heuristic = 'manhattan' if choice == '1' else 'misplaced'

puzzle = EightPuzzle(initial\_state)

print("\nInitial State:")

print\_board(puzzle.current\_state)

success, steps = puzzle.hill\_climbing(heuristic=heuristic)

print(f"Solution found: {success}")

print(f"Steps taken: {steps}")

print("Final State:")

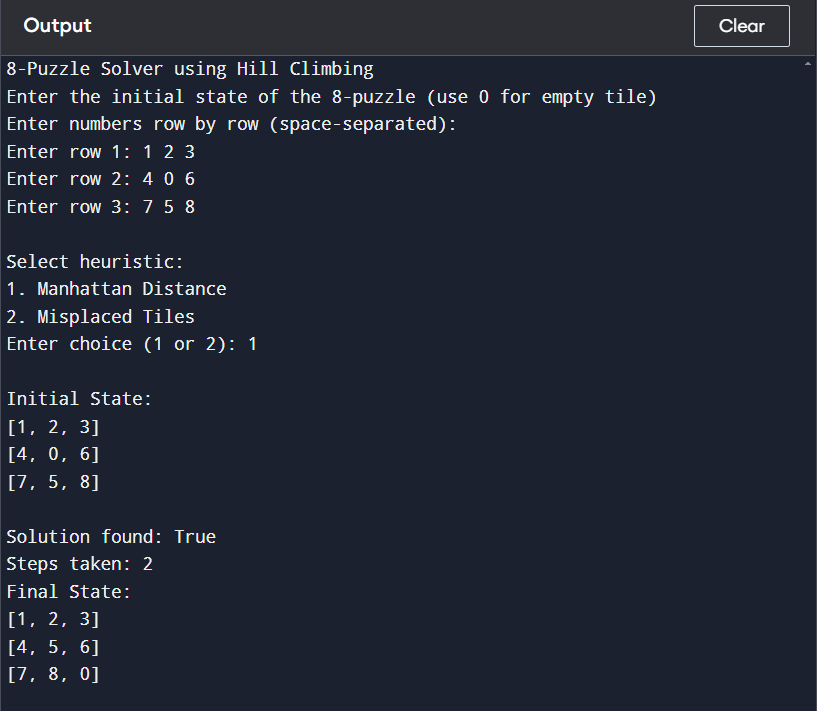
print\_board(puzzle.current\_state)

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

main()

**Output:**

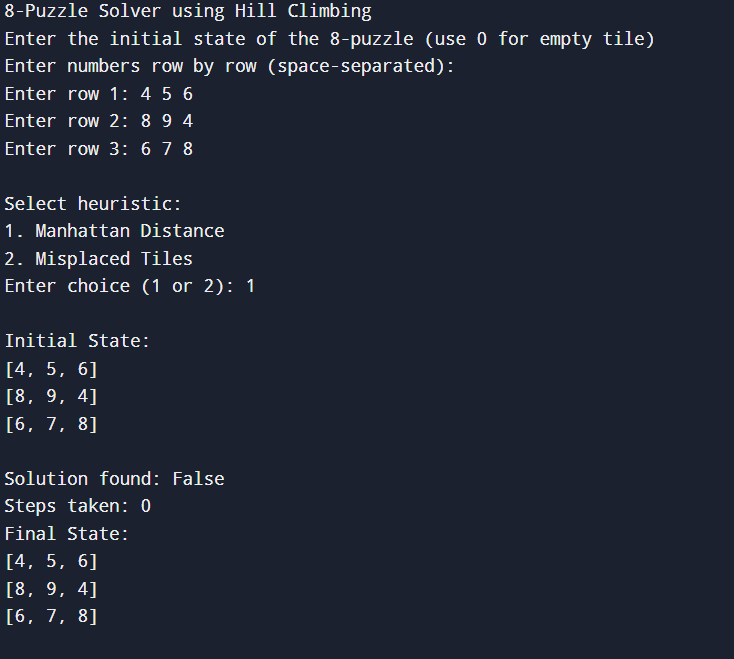
**Algorithm passes**

****

**Algorithm Fails**

**Reason:**

The algorithm works by evaluating neighboring states and selecting moves that decrease the distance to the goal state. However, in this configuration, all possible moves would temporarily increase the Manhattan distance, and since Hill Climbing doesn't allow for "worse" intermediate states that might eventually lead to a better solution, it fails immediately without taking any steps. This highlights a fundamental limitation of the Hill Climbing approach - its inability to escape local maxima by accepting temporarily suboptimal moves that could ultimately lead to the global optimum.

****

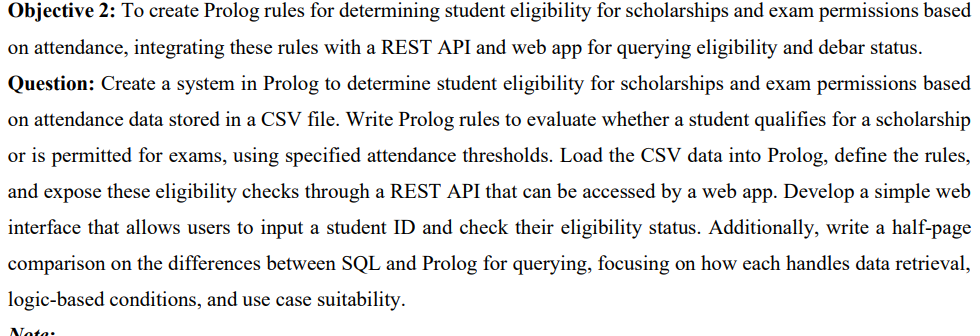
**Limitations:**

a) Local Maxima:

* The algorithm can get stuck in local maxima where no immediate moves improve the situation
* This happens when all neighboring states have worse heuristic values
* The implementation detects this when no better moves are available

b) Plateaus:

* Areas in the state space where neighboring states have equal heuristic values
* The algorithm might wander without making progress
* Current implementation might stop at plateaus since it requires strict improvement



CSV file-

data.csv

Student\_ID,Name,Attendance\_percentage,CGPA

38,Lavanya,80,9.2

25,Ram,60,8.5

31,Nikhil,90,9.5

42,Shreya,70,6.8

Setup the Prolog Environment and Load the CSV

:- use\_module(library(csv)).

:- use\_module(library(http/thread\_httpd)).

:- use\_module(library(http/http\_dispatch)).

:- use\_module(library(http/http\_json)).

Code.pl

load\_student\_data :-

csv\_read\_file("data.csv", Rows, [functor(student), arity(4)]),

maplist(assert, Rows).

eligible\_for\_scholarship(Student\_ID) :-

student(Student\_ID, \_, Attendance\_percentage, CGPA),

Attendance\_percentage >= 75,

CGPA >= 9.0.

permitted\_for\_exam(Student\_ID) :-

student(Student\_ID, \_, Attendance\_percentage, \_),

Attendance\_percentage >= 75.

start\_server(Port) :-

http\_server(http\_dispatch, [port(Port)]).

:- http\_handler('/eligibility', eligibility\_handler, []).

eligibility\_handler(Request) :-

http\_parameters(Request, [id(Student\_ID, [atom])]),

eligibility\_status(Student\_ID, Status),

reply\_json\_dict(Status).

eligibility\_status(Student\_ID, Status) :-

( eligible\_for\_scholarship(Student\_ID)

-> Scholarship = "Eligible"

; Scholarship = "Not Eligible"

),

( permitted\_for\_exam(Student\_ID)

-> Exam = "Permitted"

; Exam = "Debarred"

),

Status = \_{student\_id: Student\_ID, scholarship: Scholarship, exam\_permission: Exam}.

**Web page:**

HTML file

Index.html

<!DOCTYPE html>

<html>

<head>

<title>Student Eligibility Checker</title>

</head>

<body>

<h1>Check Student Eligibility</h1>

<form id="eligibilityForm">

<label for="studentId">Student ID:</label>

<input type="text" id="studentId" name="studentId" required>

<button type="submit">Check</button>

</form>

<div id="result"></div>

<script>

document.getElementById('eligibilityForm').addEventListener('submit', async (event) => {

event.preventDefault();

const studentId = document.getElementById('studentId').value;

try {

const response = await fetch(`/eligibility?id=${studentId}`);

const data = await response.json();

document.getElementById('result').innerHTML = `

<p>Student ID: ${data.student\_id}</p>

<p>Scholarship: ${data.scholarship}</p>

<p>Exam Permission: ${data.exam\_permission}</p>

`;

} catch (error) {

document.getElementById('result').innerHTML = `<p>Error fetching data.</p>`;

}

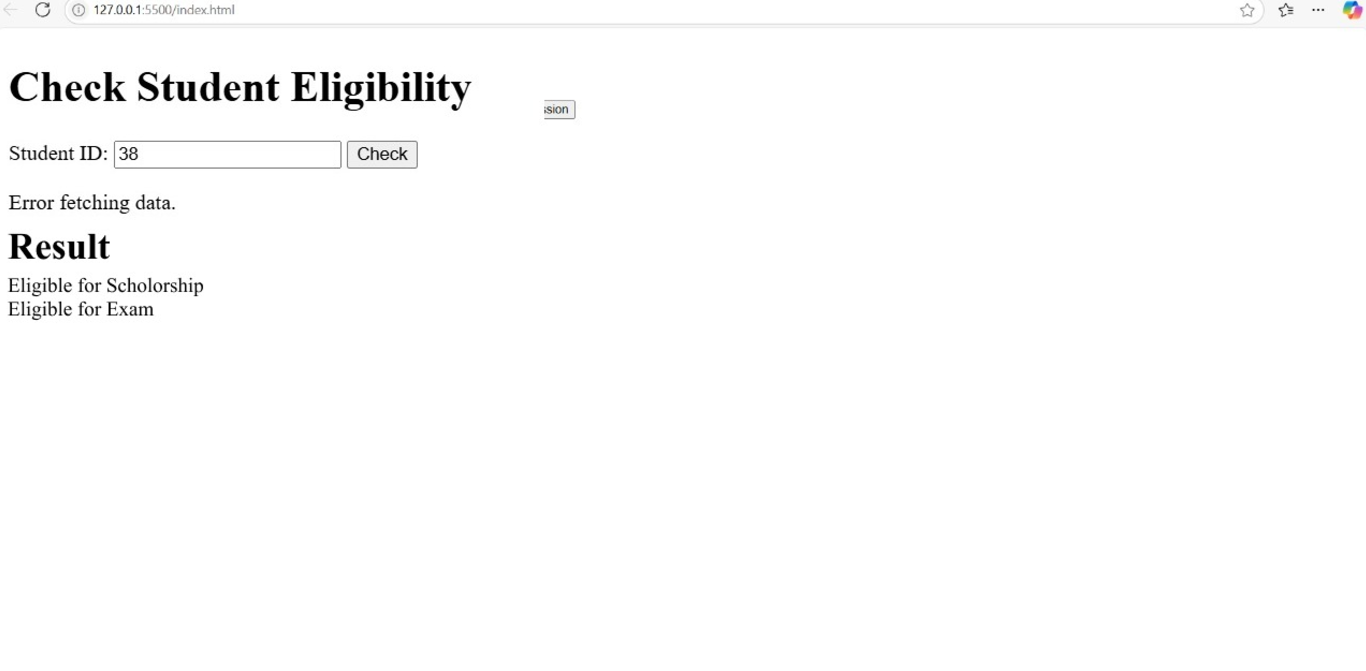
});

</script>

</body>

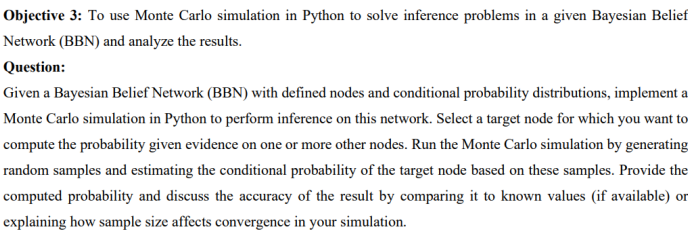
</html>

Output:



**Comparison: SQL vs Prolog**

| **Aspect** | **SQL** | **Prolog** |
| --- | --- | --- |
| Query Type | Data retrieval and aggregation. | Logic-based conditions and reasoning. |
| Data Handling | Relational; tables with structured data. | Facts and rules for logical inference. |
| Suitability | Ideal for transactional and batch queries. | Best for reasoning over rules and logic. |
| Complex Queries | Uses joins and subqueries. | Uses recursive rules and backtracking. |
| Performance | Optimized for large datasets. | Efficient for logical constraints, not large datasets. |

****

**Bayesian Belief Networks**

**Definition:** Graphical models where nodes are random variables and edges show conditional dependencies.  
**Features:**

* Each node has a Conditional Probability Distribution (CPD).
* Captures causal relationships between variables.  
  **Example:** Weather Network
* **Variables:** Cloudy, Rain, Sprinkler, Wet Grass.
* **Edges:** Show causal links, e.g., Cloudy → Rain.

**Monte Carlo Simulation**

**Definition:** A method using random sampling to approximate probabilities or numerical results.  
**Steps:**

1. Sample random values based on CPDs.
2. Count outcomes to estimate probabilities.  
   **Accuracy:** Improves with more samples.

**Applications**

* Weather prediction.
* Decision-making under uncertainty.
* Diagnosing medical conditions.

**Strengths and Limitations**

**Strengths:**

* Handles uncertainty and incomplete data efficiently.
* Useful for complex problems where exact solutions are hard.  
  **Limitations:**
* Needs many samples for accuracy.
* Relies on correct CPDs.

Code:

import numpy as np

P\_Cloudy = 0.5

P\_Sprinkler\_given\_Cloudy = {True: 0.1, False: 0.5}

P\_Rain\_given\_Cloudy = {True: 0.8, False: 0.2}

P\_WetGrass\_given\_Sprinkler\_Rain = {

(True, True): 0.99,

(True, False): 0.90,

(False, True): 0.80,

(False, False): 0.00

}

def monte\_carlo\_simulation(num\_samples=10000):

count\_sprinkler\_given\_rain = 0

count\_rain = 0

for \_ in range(num\_samples):

# Simulate events

cloudy = np.random.rand() < P\_Cloudy

sprinkler = np.random.rand() < P\_Sprinkler\_given\_Cloudy[cloudy]

rain = np.random.rand() < P\_Rain\_given\_Cloudy[cloudy]

wet\_grass = np.random.rand() < P\_WetGrass\_given\_Sprinkler\_Rain[(sprinkler, rain)]

if rain:

count\_rain += 1

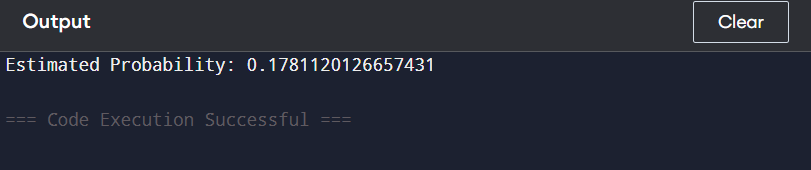
if sprinkler:

count\_sprinkler\_given\_rain += 1

return count\_sprinkler\_given\_rain / count\_rain if count\_rain > 0 else 0

print(f"Estimated Probability: {monte\_carlo\_simulation()}")

Output:



How sample size affects:

In Monte Carlo methods, the law of large numbers ensures that as the number of samples increases, the estimated probability converges toward the true probability. However, this convergence is stochastic, meaning that the accuracy improves in a probabilistic sense.  
The more samples you generate, the better the approximation of the target probability becomes. For example, with a small sample size, the estimate might be significantly biased due to random fluctuations. However, as the sample size grows, these fluctuations average out, and the result approaches the true value.