EXPERIMENT 2.2

Student Name: Samuel **UID: 24MAI10018**

Branch: AIML **Section: 24MAI-1**

Semester: 2 **Date : / /2025**

Subject Name: SC Lab **Subject Code: 24CSH–668**

AIM: Implementation of Simple Genetic Application- Match word Problem

SOFTWARE USED :JUPYTER Notebook

THEORY: Genetic algorithm is a search technique used in computing to find true or approximate solutions to approximate solutions to optimization & search problems.
Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved.
Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chances they have to reproduce.
This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

Basic Working of Genetic Algorithm Genetic Algorithms iteratively refine candidate solutions over multiple generations using genetic operators:

1. **Selection**: Choosing the fittest individuals from the population for reproduction.
2. **Crossover:** Combining genetic material from two parent chromosomes to produce offspring.
3. **Mutation:** Introducing random modifications to maintain genetic diversity.

This process continues until an optimal or near-optimal solution is found based on a defined fitness function.

Mathematical Representation of Genetic Algorithm
A Genetic Algorithm works by evaluating a fitness function:

F(x) = Evaluation of fitness based on problem constraints

The GA follows these steps:
1. Initialize Population: Create an initial set of candidate solutions.
2. Evaluate Fitness: Compute the fitness of each chromosome.
3. Selection: Choose individuals with higher fitness for reproduction.
4. Crossover: Combine selected chromosomes to create new offspring.
5. Mutation: Introduce random changes in offspring to ensure genetic diversity.
6. Termination: Stop when the best solution reaches a predefined quality threshold or after a set number of generations.

Key Components of Genetic Algorithm

1. **Population**: A set of candidate solutions (chromosomes). Each chromosome encodes a potential solution to the problem.
2. **Fitness Function:** Evaluates how close a chromosome is to the optimal solution.

3. Selection: Mechanism for choosing the fittest individuals for reproduction. Common methods include roulette wheel selection and tournament selection.
4. Crossover (Recombination): Genetic recombination to produce new offspring by exchanging segments of parent chromosomes.
5. Mutation: Randomly alters genes in offspring to introduce new genetic variations.
6. Termination Condition: The algorithm stops when a satisfactory solution is found or a maximum number of generations is reached.

Algorithm:
Match Word Finding Algorithm:

Step 1: Select the word to be guessed
This value is taken through user input.

Step 2: Initialize the population User inputs the population.

Step 3: Evaluate the population
Fitness is assigned based on number of correct letters in correct place.

Step 4: Select breeding population
Selection is done on the basis of fitness.

Step 5: Create new population
Population is created by using uniform crossover between breeding populations.

Step 6: Check for stopping condition
Here maximum fitness value in population is checked. If it is 60%.

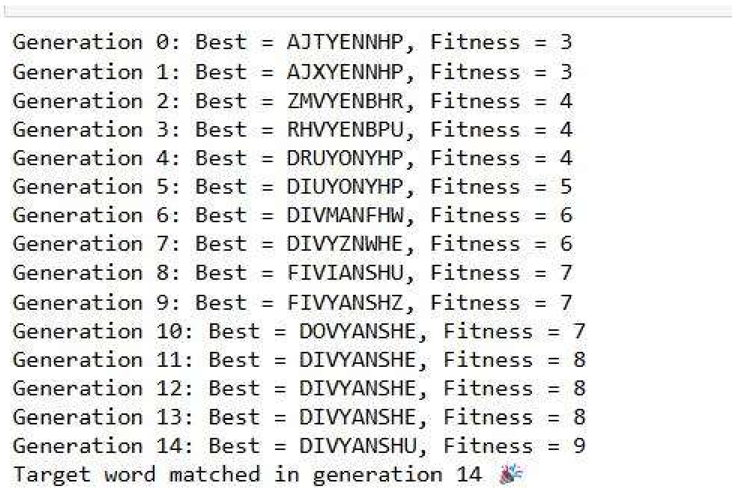
Step 7: If stopping condition is not true, go to Step3;
Else return the offspring with highest fitness value.

SOURCE CODE

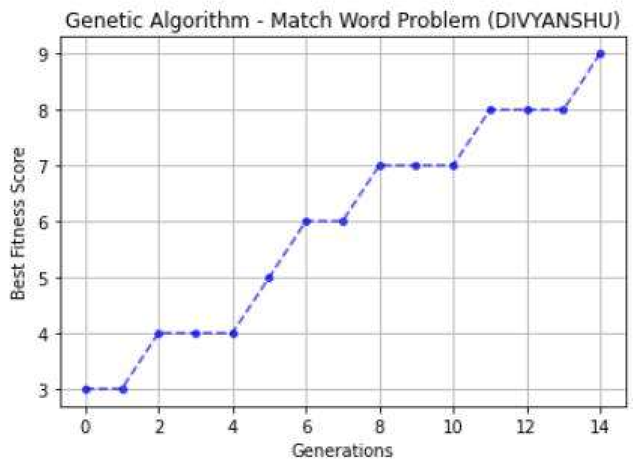
```
import numpy as np
import random
import string
import matplotlib.pyplot as plt

# Target word to match
target_word = "DIVYANSHU"

# GA Hyperparameters
population_size = 150 # Increased population for diversity
mutation_rate = 0.08 # Slightly reduced mutation rate
max_generations = 1000  # Maximum number of generations

# Generate a random word of given length
def random_word(length):
   return ''.join(random.choices(string.ascii_uppercase, k=length))

# Fitness function: Higher score for more matching characters
def fitness(word):
   return sum(1 for a, b in zip(word, target_word) if a == b)

# Selection: Keep the best and some random lower-fitness
individuals def selection(population):
   sorted_pop = sorted(population, key=fitness,
   reverse=True) top_half = sorted_pop[:population_size // 2]
   random_sample = random.sample(sorted_pop[population_size // 2:], population_size //
   10) return top_half + random_sample # Keeping some weaker candidates for diversity

# Two-point Crossover
def crossover(parent1, parent2):
   point1, point2 = sorted(random.sample(range(1, len(target_word)), 2))
   return parent1[:point1] + parent2[point1:point2] + parent1[point2:]

# Mutation: Random character change with a
probability def mutate(word):
   if random.random() < (mutation_rate + random.uniform(-0.02, 0.02)): # Adding slight randomness
      index = random.randint(0, len(word) - 1)
```

```
        word = word[:index] + random.choice(string.ascii_uppercase) + word[index +
    1:] return word

# Initialize Population
population = [random_word(len(target_word)) for _ in range(population_size)]
fitness_history = []
generation = 0

# Genetic Algorithm loop
while generation < max_generations:
    population = selection(population)
    new_population = []

    # Generate new offspring
    for _ in range(population_size):
        p1, p2 = random.sample(population, 2)
        offspring = mutate(crossover(p1, p2))
        new_population.append(offspring)

    population = new_population
    best_fit = max(population, key=fitness)
    fitness_history.append(fitness(best_fit))

    print(f"Generation {generation}: Best = {best_fit}, Fitness = {fitness(best_fit)}")

    if best_fit == target_word:
        print(f"Target word matched in generation {generation}      ")
        break

    generation += 1

# Plot fitness progression
plt.plot(fitness_history, linestyle='--', marker='o', markersize=4, color='b', alpha=0.7)
plt.xlabel("Generations")
plt.ylabel("Best Fitness Score")
plt.title("Genetic Algorithm - Match Word Problem (DIVYANSHU)")
plt.grid(True)
plt.show()
```

SCREENSHOT OF OUTPUT:



```
Generation 0: Best = AJTYENNHP, Fitness = 3
Generation 1: Best = AJXYENNHP, Fitness = 3
Generation 2: Best = ZMVYENBHR, Fitness = 4
Generation 3: Best = RHVYENBPU, Fitness = 4
Generation 4: Best = DRUYONYHP, Fitness = 4
Generation 5: Best = DIUYONYHP, Fitness = 5
Generation 6: Best = DIVMANFHW, Fitness = 6
Generation 7: Best = DIVYZNWHE, Fitness = 6
Generation 8: Best = FIVIANSHU, Fitness = 7
Generation 9: Best = FIVYANSHZ, Fitness = 7
Generation 10: Best = DOVYANSHE, Fitness = 7
Generation 11: Best = DIVYANSHE, Fitness = 8
Generation 12: Best = DIVYANSHE, Fitness = 8
Generation 13: Best = DIVYANSHE, Fitness = 8
Generation 14: Best = DIVYANSHU, Fitness = 9
Target word matched in generation 14 🎉
```



LEARNING OUTCOME:

1. Creating an understanding about the way the GA is used and the domain of application.
2. To appreciate the use of various GA operators in solving different types of GA problems.
3. Match the industry requirements in the domains of Programming and Networking with the required management skills.