Program 4 - Cuckoo Search for Optimizing a Mathematical Function :

Design and implement a Cuckoo Search (CS) algorithm in Python to find the global maximum of a given mathematical function. The CS algorithm will mimic the brood parasitism behavior of cuckoo birds, using Lévy flights to explore the solution space and discover the optimal solution.

Algorithm :

1. **Define the Problem**: Use f(x)=−x2+7x+10f(x) = -x^2 + 7x + 10f(x)=−x2+7x+10 as the objective function.
2. **Initialize Parameters**:
   - Number of nests NNN
   - Probability of discovery pap_apa
   - Number of iterations III
3. **Initialize Population**: Randomly generate initial positions of nests within the search space.
4. **Evaluate Fitness**:
   - Calculate f(x)f(x)f(x) for each nest to evaluate its fitness.
5. **Generate New Solutions**:
   - Use Lévy flights to create new solutions for each nest.
6. **Abandon Worst Nests**:
   - With probability pap_apa, replace the worst-performing nests with random solutions.
7. **Iterate**:
   - Repeat the evaluation, updating, and replacement process for III iterations.
8. **Output the Best Solution**:
   - Track and output the nest position with the highest fitness.

code:

```
import numpy as np

# Define the fitness function
def fitness_function(x):
    return -x**2 + 7*x + 10

# Lévy flight function
def levy_flight(Lambda=1.5):
    sigma = (np.math.gamma(1 + Lambda) * np.sin(np.pi * Lambda / 2) /
             (np.math.gamma((1 + Lambda) / 2) * Lambda * 2**((Lambda - 1) /
```

```
2)))**(1 / Lambda)
    u = np.random.normal(0, sigma, 1)
    v = np.random.normal(0, 1, 1)
    step = u / abs(v)**(1 / Lambda)
    return step

# Cuckoo Search implementation
def cuckoo_search():
    # Input parameters
    num_nests = int(input("Enter number of nests: "))
    prob_discovery = float(input("Enter probability of discovery (p_a): "))
    num_iterations = int(input("Enter number of iterations: "))
    x_min = float(input("Enter minimum value of x: "))
    x_max = float(input("Enter maximum value of x: "))

    # Initialize nests
    nests = np.random.uniform(x_min, x_max, num_nests)
    fitness = np.array([fitness_function(x) for x in nests])

    best_nest = nests[np.argmax(fitness)]
    best_fitness = max(fitness)

    for _ in range(num_iterations):
        # Generate new solutions using Lévy flights
        for i in range(num_nests):
            step_size = levy_flight()
            nests[i] += step_size
            nests[i] = np.clip(nests[i], x_min, x_max)

        # Evaluate new fitness
        new_fitness = np.array([fitness_function(x) for x in nests])

        # Update nests with better solutions
        for i in range(num_nests):
            if new_fitness[i] > fitness[i]:
                fitness[i] = new_fitness[i]
                nests[i] = nests[i]

        # Abandon a fraction of the worst nests
        abandon_count = int(prob_discovery * num_nests)
        worst_indices = np.argsort(fitness)[:abandon_count]
```

```
        for idx in worst_indices:
            nests[idx] = np.random.uniform(x_min, x_max)
            fitness[idx] = fitness_function(nests[idx])

        # Update best solution
        current_best_idx = np.argmax(fitness)
        if fitness[current_best_idx] > best_fitness:
            best_fitness = fitness[current_best_idx]
            best_nest = nests[current_best_idx]

    return best_nest, best_fitness

# Run the Cuckoo Search algorithm
best_solution, best_value = cuckoo_search()
print(f"Best Solution: {best_solution}, Fitness: {best_value}")
print("Name-pooja Gaikwad(1BM22CS194)")
```

Output:

```
Enter number of nests: 10
Enter probability of discovery (p_a): 0.25
Enter number of iterations: 100
Enter minimum value of x: -10
Enter maximum value of x: 10
Best Solution: 3.506038131134606, Fitness: 22.249963540972402
Name-pooja Gaikwad(1BM22CS194)
```

Observation:

Page No.
Date

④ Cuckoo Search Algo

1. Start

2. Initialize parameters :
   n ← no. of nests
   max-iter ← max. no. of iter
   Initialise population of n nests

3. nests = Random nests (n)
   fitness = evaluate fitness (nests)

   While termination-cond not met :
   for each nest is nests :
   new-nest = Generate sol (nest)

   [Generate new sol. using levy flight)

   new-nests = nest + step-size * levy (β)
   new-fitness = evaluate fitness (new-nests)

   if new-fitness > fitness
   nests (nest) = new-nests (nest)
   fitness (nest) = new-fitness

4. Replace worst nests

   nests = Replace (nests, fitness)
   best-nest = Best sol (nest, fitness)