# Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Program: Bachelor of Science in Computer Science and Engineering

Course No: CSE 4108

Course Title: Artificial Intelligence Lab

Assignment No: 03

Date of Submission: 29/1/2022

Submitted to:

Mr. Faisal Muhammad Shah
Associate Professor, Department of CSE, AUST.

Mr. Md. Siam Ansary
Lecturer, Department of CSE, AUST.

Submitted by,

Name: Atanu Kumar Saha

Student ID: 17.02.04.003

**Question 1**: Implement Genetic Algorithm for 8 Queens problem using Python. For the mutation step, use Swap mutation.

**Solution:**

**Python Code:**

```
import sys
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.cm as cm

def checkFitness(pop):
    fit = np.zeros((pop[:, 1].size, 1))
    for index, solution in enumerate(pop):
        for ia, a in enumerate(solution, start=1):
            for ib, b in enumerate(solution[ia:(len(solution))], start=ia
+ 1):
                if abs(a - b) == abs(ia - ib):
                    fit[index, 0] = fit[index, 0] + 1
    return fit

def order_crossover(p1, p2, size):
    def fillGene(f, p):
        for ia, a in enumerate(p):
            if a not in f:
                for ib, b in enumerate(f):
                    if b == 0:
                        f[ib] = a
                        break
        return f
    f1 = np.zeros(len(p1))
    f2 = np.zeros(len(p2))
    c = random.randint(0, (len(p1) - size))
    f1[c:c + size] = p1[c:c + size]
    f2[c:c + size] = p2[c:c + size]
    f1 = fillGene(f1, p2)
    f2 = fillGene(f2, p1)
    offsprings = np.vstack([f1, f2])
    return offsprings
```

```python
def selection(pop, p_sel):
    sel_pool = np.random.permutation(pop[:, 1].size)[0:int(round(pop[:, 1]
.size * p_sel))]
    bestSol = pop[sel_pool[0], :]
    for sol in sel_pool[1:len(sel_pool)]:
        if pop[sol, len(bestSol) - 1] < bestSol[len(bestSol) - 1]:
            bestSol = pop[sol, :]
    return bestSol

def swap_mutation(child, numberOfSwaps):
    for i in range(numberOfSwaps):
        swapGenesPairs = np.random.choice(len(child), 2, replace=False)
        a = child[swapGenesPairs[0]]
        b = child[swapGenesPairs[1]]
        child[swapGenesPairs[0]] = b
        child[swapGenesPairs[1]] = a
    return child

def plotCheckBoard(sol):
    def checkerboard(shape):
        return np.indices(shape).sum(axis=0) % 2
    sol = sol - 1
    size = len(sol)
    color = 0.5
    board = checkerboard((size, size)).astype('float64')
    for i in range(size):
        board[i, int(sol[i])] = color
        fig, ax = plt.subplots()
        ax.imshow(board, cmap=plt.cm.CMRmap, interpolation='nearest')
        plt.show()

npop = 15
size = 8
ox_size = 2
generation = 10000

p_sel = 1

p_m = 1
numberOfSwaps = 2

pop = np.zeros((npop, size))
```

```
for i in range(npop):
    pop[i, :] = np.random.permutation(size) + 1
fit = checkFitness(pop)

pop = np.hstack((pop, fit))
print("No ", " Initial Population", " ", " Fitness Score")
for i in range(npop):
    print(i, " ", pop[i, 0:8], " ", pop[i, 8:9])
print("Sorted by Fitness Score (Ascending Order)")
sorted_list = sorted(pop, key=lambda item: (item[8], item[7]))
print(*sorted_list, sep="\n")
meanFit = np.zeros(generation)

for gen in range(generation):
    parents = [selection(pop, p_sel), selection(pop, p_sel)]
    print("Parents: ")
    print(parents)
    offsprings = order_crossover(parents[0][0:size], parents[1][0:size], o
x_size)
    print("Children: ")
    print(offsprings)

    for child in range(len(offsprings)):
        r_m = round(random.random(), 2)
        if r_m <= p_m:
            offsprings[child] = swap_mutation(offsprings[child], numberOfS
waps)
    fitOff = checkFitness(offsprings)
    offsprings = np.hstack((offsprings, fitOff))

    pop = np.vstack([pop, offsprings])
    pop = pop[pop[:, size].argsort()][0:npop, :]
    print("Crossover with Fitness: ")
    print(pop)
    print("Generation ", gen + 1, " Done")
    comparison = pop[np.argmin(pop[:, size]), :]
    a = comparison[size:size + 1]
    if a == 0:
        break
    else:
        continue

bestSol = pop[np.argmin(pop[:, size]), :]
```

```
print(bestSol[0:size])
print(f"Best Solution have: {bestSol[size]} Conflict(s)")
plotCheckBoard(bestSol[0:size])
```

**Question 2**: Implement A* search algorithm using Python.

**Solution:**

**Python Code:**

```
tree = {'S': [['A', 1], ['B', 5], ['C', 8]],
        'A': [['S', 1], ['D', 3], ['E', 7], ['G', 9]],
        'B': [['S', 5], ['G', 4]],
        'C': [['S', 8], ['G', 5]],
        'D': [['A', 3]],
        'E': [['A', 7]]}



heuristic = {'S': 8, 'A': 8, 'B': 4, 'C': 3, 'D': 5000, 'E': 5000, 'G': 0}


cost = {'S': 0} # total cost for nodes visited

def AStarSearch():
    global tree,heuristic
    closed=[]
    opened=[['S',8]]

    '''find the visited nodes'''
    while True:
        fn = [i[1] for i in opened]  # fn = f(n) = g(n) + h(n)
        chosen_index = fn.index(min(fn))
        node = opened[chosen_index][0]  # current node
        closed.append(opened[chosen_index])
        del opened[chosen_index]
        if closed[-
1][0] == 'G':  # break the loop if node G has been found
            break
        for item in tree[node]:
            if item[0] in [closed_item[0] for closed_item in closed]:
                continue
            cost.update({item[0]: cost[node] + item[1]})  # add nodes to c
ost dictionary
            fn_node = cost[node] + heuristic[item[0]] + item[1]  # calcula
te f(n) of current node
            temp = [item[0], fn_node]
            opened.append(temp)  # store f(n) of current node in array ope
ned


    '''find optimal path'''
    trace_node = 'G'  # correct optimal tracing node, initialize as node G
    optimal_sequence = ['G']  # optimal node sequence
    for i in range(len(closed) - 2, -1, -1):
        check_node = closed[i][0]  # current node
        if trace_node in [children[0] for children in tree[check_node]]:
            children_costs = [temp[1] for temp in tree[check_node]]
            children_nodes = [temp[0] for temp in tree[check_node]]
            '''check whether h(s) + g(s) = f(s). If so, append current nod
e to optimal sequence
             change the correct optimal tracing node to current node'''
            if cost[check_node] + children_costs[children_nodes.index(trac
e_node)] == cost[trace_node]:
                optimal_sequence.append(check_node)
                trace_node = check_node
                optimal_sequence.reverse()  # reverse the optimal sequence
                return closed, optimal_sequence

if __name__ == '__main__':
    visited_nodes, optimal_nodes = AStarSearch()
    print('visited nodes: ' + str(visited_nodes))
    print('optimal nodes sequence: ' + str(optimal_nodes))
```

```
visited nodes: [['S', 8], ['A', 9], ['B', 9], ['G', 9]]
optimal nodes sequence: ['B', 'G']
```