

# 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: 30 / 01 / 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: Nusrat Jahan

Student ID: 180104020

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

**Solution:**

Python Code:

```
import random
```

```
def view(li, index):
    print()
    print(f"Solution number {index + 1}: ", end='')
    print(li)
    print()

    for i in range(8):
        x = li[i] - 1
        for j in range(8):
            if j == x:
                print('[Q]', end='')
            else:
                print('[ ]', end='')
        print()

    print()

def getHuristic(instance):
    huristic = []
    for i in range(len(instance)):
        j = i - 1
        huristic.append(0)
        while j >= 0:
            if instance[i] == instance[j] or (abs(instance[i] -
instance[j]) == abs(i - j)):
                huristic[i] += 1
            j -= 1
        j = i + 1
        while j < len(instance):
            if instance[i] == instance[j] or (abs(instance[i] -
instance[j]) == abs(i - j)):
                huristic[i] += 1
            j += 1
    return huristic
```

```

def getFitness(instance):
    clashes = 0
    for i in range(len(instance) - 1):
        for j in range(i + 1, len(instance)):
            if instance[i] == instance[j]:
                clashes += 1
    for i in range(len(instance) - 1):
        for j in range(i + 1, len(instance)):
            if abs(instance[j] - instance[i]) == abs(j - i):
                clashes += 1
    return 28 - clashes

def buildKid(instance1, instance2, crossOver):
    newInstance = []
    for i in range(crossOver):
        newInstance.append(instance1[random.randint(0, 7)])
    for i in range(crossOver, 8):
        newInstance.append(instance2[random.randint(0, 7)])
    return newInstance

def changeChilds(co):
    global father, mother, child1, child2, crossover
    crossover = co
    child1 = buildKid(father, mother, crossover)
    child2 = buildKid(mother, father, crossover)

def changeChromosome(li):
    global crossover, father, mother
    newchange = -1
    while newchange != 0:
        newchange = 0
        tmpli = li
        getHur = getHuristic(tmpli)
        index = getHur.index(max(getHur))
        maxFitness = getFitness(tmpli)
        for i in range(1, 9):
            tmpli[index] = i
            if getFitness(tmpli) > maxFitness:
                maxFitness = getFitness(tmpli)
                newchange = i
            tmpli = li
    if newchange == 0:

```

```

        for i in range(len(li) - 1):
            for j in range(i + 1, len(li)):
                if li[i] == li[j]:
                    li[j] = random.randint(1, 8)
    else:
        li[index] = newchange

if __name__ == "__main__":
    numberOfSolutions = int(input())

    solutions = []
    crossover = 4
    while len(solutions) < numberOfSolutions:
        father = []
        mother = []
        for i in range(8):
            father.append(random.randint(1, 8))
            mother.append(random.randint(1, 8))
        fitnessFather = getFitness(father)
        fitnessMother = getFitness(mother)
        while fitnessFather != 28 and fitnessMother != 28:
            changeChilds(crossover)
            changeChromosome(child1)
            changeChromosome(child2)
            fitnessFather = getFitness(child1)
            fitnessMother = getFitness(child2)
            father = child1
            mother = child2
            print(father)
            print(mother)
        if getFitness(father) == 28:
            if father not in solutions:
                solutions.append(father)
        else:
            if mother not in solutions:
                solutions.append(mother)

    for i in range(len(solutions)):
        view(solutions[i], i)

```

```

1      import random
2
3
4      def view(li, index):
5          print()
6          print(f"Solution number {index + 1}: ", end='')
7          print(li)
8          print()
9
10     for i in range(8):
11         x = li[i] - 1
12         for j in range(8):
13             if j == x:
14                 print('[Q]', end='')
15             else:
16                 print('[ ]', end='')
17         print()
18
19     print()
20
21
--
22     def getHuristic(instance):
23         huristic = []
24         for i in range(len(instance)):
25             j = i - 1
26             huristic.append(0)
27             while j >= 0:
28                 if instance[i] == instance[j] or (abs(instance[i] - instance[j]) == abs(i - j)):
29                     huristic[i] += 1
30                 j -= 1
31             j = i + 1
32             while j < len(instance):
33                 if instance[i] == instance[j] or (abs(instance[i] - instance[j]) == abs(i - j)):
34                     huristic[i] += 1
35                 j += 1
36         return huristic
37
38
39     def getFitness(instance):
40         clashes = 0
41         for i in range(len(instance) - 1):
42             for j in range(i + 1, len(instance)):
43                 if instance[i] == instance[j]:
44                     clashes += 1
45         for i in range(len(instance) - 1):
46             for j in range(i + 1, len(instance)):
47                 if abs(instance[j] - instance[i]) == abs(j - i):

```

```

47         if abs(instance[j] - instance[i]) == abs(j - i):
48             clashes += 1
49     return 28 - clashes
50
51
52 def buildKid(instance1, instance2, crossover):
53     newInstance = []
54     for i in range(crossover):
55         newInstance.append(instance1[random.randint(0, 7)])
56     for i in range(crossover, 8):
57         newInstance.append(instance2[random.randint(0, 7)])
58     return newInstance
59
60
61 def changeChilds(co):
62     global father, mother, child1, child2, crossover
63     crossover = co
64     child1 = buildKid(father, mother, crossover)
65     child2 = buildKid(mother, father, crossover)

```

```

68 def changeChromosome(li):
69     global crossover, father, mother
70     newchange = -1
71     while newchange != 0:
72         newchange = 0
73         tmpli = li
74         getHur = getHuristic(tmpli)
75         index = getHur.index(max(getHur))
76         maxFitness = getFitness(tmpli)
77         for i in range(1, 9):
78             tmpli[index] = i
79             if getFitness(tmpli) > maxFitness:
80                 maxFitness = getFitness(tmpli)
81                 newchange = i
82                 tmpli = li
83             if newchange == 0:
84                 for i in range(len(li) - 1):
85                     for j in range(i + 1, len(li)):
86                         if li[i] == li[j]:
87                             li[j] = random.randint(1, 8)
88             else:
89                 li[index] = newchange
90
91
92 if __name__ == "__main__":
93     numberOfSolutions = int(input())
94
95     solutions = []
96     crossover = 4
97     while len(solutions) < numberOfSolutions:
98         father = []
99         mother = []
100         for i in range(8):
101             father.append(random.randint(1, 8))
102             mother.append(random.randint(1, 8))
103         fitnessFather = getFitness(father)
104         fitnessMother = getFitness(mother)
105         while fitnessFather != 28 and fitnessMother != 28:
106             changeChilds(crossover)
107             changeChromosome(child1)
108             changeChromosome(child2)
109             fitnessFather = getFitness(child1)
110             fitnessMother = getFitness(child2)
111             father = child1

```

```
111         father = child1
112         mother = child2
113         print(father)
114         print(mother)
115     if getFitness(father) == 28:
116         if father not in solutions:
117             solutions.append(father)
118     else:
119         if mother not in solutions:
120             solutions.append(mother)
121
122     for i in range(len(solutions)):
123         view(solutions[i], i)
124
```



Solution number 1: [4, 8, 5, 3, 1, 7, 2, 6]

```
[ ][ ][ ][Q][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q][ ]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ][ ]
```

Solution number 2: [4, 1, 5, 8, 2, 7, 3, 6]

```
[ ][ ][ ][Q][ ][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ][ ]
```

Solution number 3: [8, 3, 1, 6, 2, 5, 7, 4]

```
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][ ][Q][ ][ ][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
```

Solution number 4: [7, 3, 8, 2, 5, 1, 6, 4]

```
[ ][ ][ ][ ][ ][ ][Q][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][Q][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][ ][Q][ ][ ][ ][ ]
```

Solution number 5: [8, 2, 5, 3, 1, 7, 4, 6]

```
[ ][ ][ ][ ][ ][ ][ ][ ][Q]
[ ][Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ][ ][ ]
```

Solution number 6: [8, 4, 1, 3, 6, 2, 7, 5]

```
[ ][ ][ ][ ][ ][ ][ ][ ][Q]
[ ][ ][ ][Q][ ][ ][ ][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ][ ][ ]
[ ][Q][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ][ ][ ]
```

Solution number 7: [1, 7, 5, 8, 2, 4, 6, 3]

```
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][Q][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
```

Solution number 8: [3, 5, 8, 4, 1, 7, 2, 6]

```
[ ][ ][Q][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][ ][ ][Q][ ][ ][ ][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ]
```

Solution number 9: [2, 7, 5, 8, 1, 4, 6, 3]

```
[ ][Q][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][Q][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
```

Solution number 10: [5, 7, 1, 3, 8, 6, 4, 2]

```
[ ][ ][ ][ ][Q][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][Q][ ]
[Q][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][Q][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][Q]
[ ][ ][ ][ ][ ][Q][ ][ ]
[ ][ ][ ][Q][ ][ ][ ][ ]
[ ][Q][ ][ ][ ][ ][ ][ ]
```

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

Python Code:

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)

    closed_set = set()

    g = {}

    parents = {}

    g[start_node] = 0

    parents[start_node] = start_node

    while len(open_set) > 0:

        n = None

        for v in open_set:

            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

                n = v

        if n == stop_node or Graph_nodes[n] == None:

            pass

        else:

            for (m, weight) in get_neighbors(n):

                if m not in open_set and m not in closed_set:

                    open_set.add(m)

                    parents[m] = n

                    g[m] = g[n] + weight

            else:
```

```

        if g[m] > g[n] + weight:
            g[m] = g[n] + weight
            parents[m] = n
            if m in closed_set:
                closed_set.remove(m)
            open_set.add(m)

    if n == None:
        print('Path does not exist!')
        return None

    if n == stop_node:
        path = []

        while parents[n] != n:
            path.append(n)
            n = parents[n]

        path.append(start_node)
        path.reverse()
        print('Path found: {}'.format(path))
        return path

    open_set.remove(n)
    closed_set.add(n)

```

```

    print('Path does not exist!')

    return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1), ('G', 9)],
    'C': None,
    'E': [('D', 6)],

```



```

'D': [('G', 1)],
}

aStarAlgo('A', 'G')

```

```

Offline3_2.py x
1  def aStarAlgo(start_node, stop_node):
2      open_set = set(start_node)
3      closed_set = set()
4      g = {}
5      parents = {}
6      g[start_node] = 0
7      parents[start_node] = start_node
8
9      while len(open_set) > 0:
10         n = None
11         for v in open_set:
12             if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
13                 n = v
14         if n == stop_node or Graph_nodes[n] == None:
15             pass
16         else:
17             for (m, weight) in get_neighbors(n):
18                 if m not in open_set and m not in closed_set:
19                     open_set.add(m)
20                     parents[m] = n
21                     g[m] = g[n] + weight
22                 else:
23                     if g[m] > g[n] + weight:
24                         g[m] = g[n] + weight
25                         parents[m] = n
26                     if m in closed_set:
27                         closed_set.remove(m)
28                     open_set.add(m)
29
30         if n == None:
31             print('Path does not exist!')
32             return None
33
34         if n == stop_node:
35             path = []
36
37             while parents[n] != n:
38                 path.append(n)

```

```

39         n = parents[n]
40
41         path.append(start_node)
42         path.reverse()
43         print('Path found: {}'.format(path))
44         return path
45
46     open_set.remove(n)
47     closed_set.add(n)
48
49     print('Path does not exist!')
50     return None
51
52 def get_neighbors(v):
53     if v in Graph_nodes:
54         return Graph_nodes[v]
55     else:
56         return None
57
58 def heuristic(n):
59     H_dist = {
60         'A': 11,
61         'B': 6,
62         'C': 99,
63         'D': 1,
64         'E': 7,
65         'G': 0,
66     }
67     return H_dist[n]
68
69 Graph_nodes = {
70     'A': [('B', 2), ('E', 3)],
71     'B': [('C', 1), ('G', 9)],
72     'C': None,
73     'E': [('D', 6)],
74     'D': [('G', 1)],
75 }
76
77 aStarAlgo('A', 'G')

```

```
C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.9.exe "D:\Python\PyCharm Community Edition 2021.3.1\  
Connected to pydev debugger (build 213.6461.77)  
Path found: ['A', 'E', 'D', 'G']  
  
Process finished with exit code 0  
|
```