Practical 1

Write a program to demonstrate FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Program:-

```
import pandas as pd
df = pd.read_csv("dataset.csv")
data = df.drop('Goes',axis='columns')
target = df.Goes

def train(concept,target):
    for i, val in enumerate(target):
        if val == "Yes":
            specific_hypothesis = concept[i].copy()
            break

    for i, val in enumerate(concept):
        if target[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",train(data,target))
```

OUTPUT

```
The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Practical 2

Write a program for Candidate Elimination algorithm for finding the consistent version space based on a given set of training data samples. The training data is read from a .CSV file.

Program:-

```
import numpy as np
import pandas as pd

df = pd.read_csv('dataset.csv')
data = df.drop('EnjoySport',axis='columns')
target = df.EnjoySport

def candidate_elimination(concepts, target):
    specific_h = concepts[0].copy()
    print("Initialization Of Specific_h And General_h\n")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
```

```
s_final, g_final = candidate_elimination(data, target)
print("\n\nFinal Specific_h:", s_final, sep="\n")
print("\n\nFinal General_h:", g_final, sep="\n")
```

OUTPUT

```
Initialization Of Specific_h And General_h

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']


Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Practical 3

Write a program to implement k-Nearest Neighbor algorithm using a set of training data samples.

In [58]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [59]:

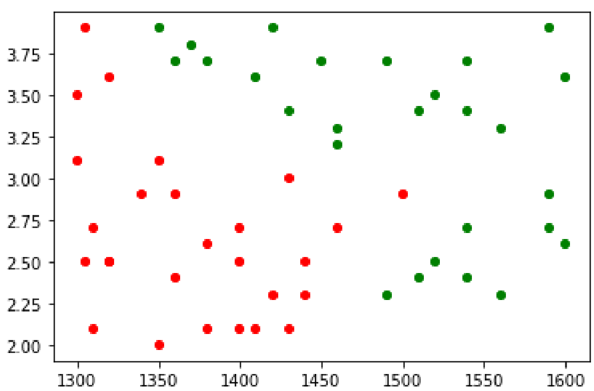
```
X = [[1590,2.9], [1540,2.7], [1600,2.6], [1590,2.7], [1520,2.5], [1540,2.4], [1560,2.3], [1490,2.3]
, [1510,2.4],
     [1350,3.9], [1360,3.7], [1370,3.8], [1380,3.7], [1410,3.6], [1420,3.9], [1430,3.4], [1450,3.7]
, [1460,3.2],
     [1590,3.9], [1540,3.7], [1600,3.6], [1490,3.7], [1520,3.5], [1540,3.4], [1560,3.3], [1460,3.3]
, [1510,3.4],
     [1340,2.9], [1360,2.4], [1320,2.5], [1380,2.6], [1400,2.1], [1320,2.5], [1310,2.7], [1410,2.1]
, [1305,2.5],
     [1460,2.7], [1500,2.9], [1300,3.5], [1320,3.6], [1400,2.7], [1300,3.1], [1350,3.1], [1360,2.9]
, [1305,3.9],
     [1430,3.0], [1440,2.3], [1440,2.5], [1380,2.1], [1430,2.1], [1400,2.5], [1420,2.3], [1310,2.1]
, [1350,2.0]]

Y = ['accepted','accepted','accepted','accepted','accepted','accepted','accepted','accepted','accep
ted',
     'accepted','accepted','accepted','accepted','accepted','accepted','accepted','accepted','accep
ted',
     'accepted','accepted','accepted','accepted','accepted','accepted','accepted','accepted','accep
ted',
     'rejected','rejected','rejected','rejected','rejected','rejected','rejected','rejected','rejec
ted',
     'rejected','rejected','rejected','rejected','rejected','rejected','rejected','rejected','rejec
ted',
     'rejected','rejected','rejected','rejected','rejected','rejected','rejected','rejected','rejec
ted']
```

In [60]:

```
for i in range(len(X)):
    if Y[i] == 'accepted':
        plt.scatter(X[i][0], X[i][1], s=10, marker='P', linewidths=2, color='green')
    else:
        plt.scatter(X[i][0], X[i][1], s=10, marker='P', linewidths=2, color='red')
plt.plot()
```

Out[60]:

```
[]
```



In [61]:

```
def most_found(array):
    list_of_words = []
    for i in range(len(array)):
        if array[i] not in list_of_words:
            list_of_words.append(array[i])

    most_counted = ''
    n_of_most_counted = None

    for i in range(len(list_of_words)):
        counted = array.count(list_of_words[i])
        if n_of_most_counted == None:
            most_counted = list_of_words[i]
            n_of_most_counted = counted
        elif n_of_most_counted < counted:
            most_counted = list_of_words[i]
            n_of_most_counted = counted
        elif n_of_most_counted == counted:
            most_counted = None

    return most_counted
```

In [62]:

```
def find_neighbors(point, data, labels, k=3):
    n_of_dimensions = len(point)

    neighbors = []
    neighbor_labels = []

    for i in range(0, k):
        nearest_neighbor_id = None
        smallest_distance = None

        for i in range(0, len(data)):
            eucledian_dist = 0
            for d in range(0, n_of_dimensions):
                dist = abs(point[d] - data[i][d])
                eucledian_dist += dist

            eucledian_dist = np.sqrt(eucledian_dist)

            if smallest_distance == None:
                smallest_distance = eucledian_dist
                nearest_neighbor_id = i
            elif smallest_distance > eucledian_dist:
                smallest_distance = eucledian_dist
                nearest_neighbor_id = i

        neighbors.append(data[nearest_neighbor_id])
        neighbor_labels.append(labels[nearest_neighbor_id])

        data.remove(data[nearest_neighbor_id])
        labels.remove(labels[nearest_neighbor_id])
    return neighbor_labels

def k_nearest_neighbor(point, data, labels, k=3):

    while True:
        neighbor_labels = find_neighbors(point, data, labels, k=k)
        label = most_found(neighbor_labels)
        if label != None:
            break
        k += 1
        if k >= len(data):
            break

    return label
```

In [63]:

```
point = [1500, 2.3]
k_nearest_neighbor(point, X, Y, k=3)
```

```
k_nearest_neighbor(point, X, Y, k=3)
```

Out[63]:

```
'accepted'
```

Practical 4

Write A Program To Implement K_Means Using Python

In [125]:

```
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

In [126]:

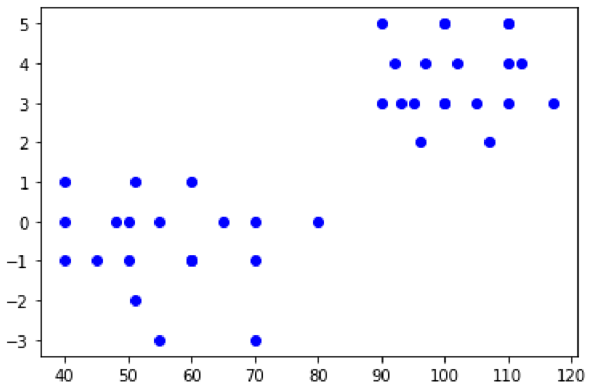
```
X = [[100,5], [90,5], [110,5], [97,4], [102,4], [112,4], [92,4], [95,3], [90,3], [100,3],
     [110,5], [100,5], [110,4], [93,3], [107,2], [117,3], [96,2], [105,3], [100,3], [110,3],
     [60,-1], [70,-1],[40,1], [70,-3], [50,-1], [80,0],[50,0],[60,-1],[60,1],[55,0],
     [40,-1], [45,-1],[40,0], [55,-3], [60,-1], [65,0],[70,0],[51,-2],[51,1],[48,0]]
```

In [127]:

```
plotx = []
ploty = []
for i in range(len(X)):
    plotx.append(X[i][0])
    ploty.append(X[i][1])
plt.plot(plotx,ploty, 'bo')
```

Out[127]:

```
[<matplotlib.lines.Line2D at 0x7f2fe00c7cd0>]
```



In [128]:

```
def random_centers(dim,k):
    centers = []
    for i in range(k):
        center = []
        for d in range(dim):
            rand = random.randint(0,100)
            center.append(rand)
        centers.append(center)
    return centers

def point_clustering(data, centers, dims, first_cluster=False):
    for point in data:
        nearest_center = 0
        nearest_center_dist = None
        for i in range(0, len(centers)):
            euclidean_dist = 0
            for d in range(0, dims):
                dist = abs(point[d] - centers[i][d])
                euclidean_dist += dist
```

```
            euclidean_dist = np.sqrt(euclidean_dist)
            if nearest_center_dist == None:
                nearest_center_dist = euclidean_dist
                nearest_center = i
            elif nearest_center_dist > euclidean_dist:
                nearest_center_dist = euclidean_dist
                nearest_center = i
        if first_cluster:
            point.append(nearest_center)
        else:
            point[-1] = nearest_center
    return data

def mean_center(data, centers, dims):
    print('centers:', centers, 'dims:', dims)
    new_centers = []
    for i in range(len(centers)):
        new_center = []
        n_of_points = 0
        total_of_points = []
        for point in data:
            if point[-1] == i:
                n_of_points += 1
                for dim in range(0,dims):
                    if dim < len(total_of_points):
                        total_of_points[dim] += point[dim]
                    else:
                        total_of_points.append(point[dim])
        if len(total_of_points) != 0:
            for dim in range(0,dims):
                print(total_of_points, dim)
                new_center.append(total_of_points[dim]/n_of_points)
            new_centers.append(new_center)
        else:
            new_centers.append(centers[i])


    return new_centers
```

In [129]:

```
def train_k_means_clustering(data, k=2, epochs=5):
    dims = len(data[0])
    print('data[0]:',data[0])
    centers = random_centers(dims,k)

    clustered_data = point_clustering(data, centers, dims, first_cluster=True)

    for i in range(epochs):
        centers = mean_center(clustered_data, centers, dims)
        clustered_data = point_clustering(data, centers, dims, first_cluster=False)

    return centers

def predict_k_means_clustering(point, centers):
    dims = len(point)
    center_dims = len(centers[0])

    if dims != center_dims:
        raise ValueError('Point given for prediction have', dims, 'dimensions but centers have', ce
nter_dims, 'dimensions')

    nearest_center = None
    nearest_dist = None

    for i in range(len(centers)):
        euclidean_dist = 0
        for dim in range(1, dims):
            dist = point[dim] - centers[i][dim]
            euclidean_dist += dist**2
        euclidean_dist = np.sqrt(euclidean_dist)
        if nearest_dist == None:
            nearest_dist = euclidean_dist
            nearest_center = i
        elif nearest_dist > euclidean_dist:
            nearest_dist = euclidean_dist
```

```
            nearest_center = i
        print('center:',i, 'dist:',euclidean_dist)

    return nearest_center
```

In [130]:

```
centers = train_k_means_clustering(X, k=2, epochs=5)
```

```
data[0]: [100, 5]
centers: [[12, 61], [89, 72]] dims: 2
[525, -6] 0
[525, -6] 1
[2631, 67] 0
[2631, 67] 1
centers: [[47.72727272727273, -0.5454545454545454], [90.72413793103448, 2.310344827586207]] dims: 
2
[1040, -12] 0
[1040, -12] 1
[2116, 73] 0
[2116, 73] 1
centers: [[54.73684210526316, -0.631578947368421], [100.76190476190476, 3.4761904761904763]] dims: 
2
[1040, -12] 0
[1040, -12] 1
[2116, 73] 0
[2116, 73] 1
centers: [[54.73684210526316, -0.631578947368421], [100.76190476190476, 3.4761904761904763]] dims: 
2
[1040, -12] 0
[1040, -12] 1
[2116, 73] 0
[2116, 73] 1
centers: [[54.73684210526316, -0.631578947368421], [100.76190476190476, 3.4761904761904763]] dims: 
2
[1040, -12] 0
[1040, -12] 1
[2116, 73] 0
[2116, 73] 1
```

In [131]:

```
print(centers)
```

```
[[54.73684210526316, -0.631578947368421], [100.76190476190476, 3.4761904761904763]]
```

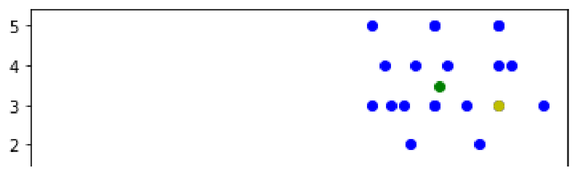
In [132]:

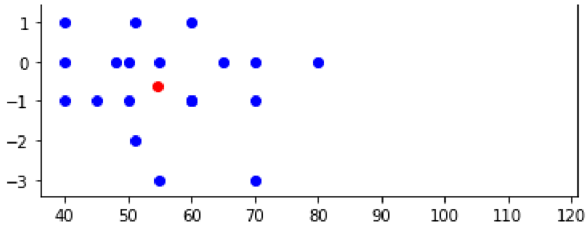
```
point = [110,3]
print(predict_k_means_clustering(point, centers))

plt.plot(plotx,ploty, 'bo', centers[0][0], centers[0][1],'ro', centers[1][0], centers[1][1], 'go', 
point[0], point[1], 'yo')
```

```
center: 0 dist: 3.6315789473684212
center: 1 dist: 0.4761904761904763
1
```

Out[132]:

```
[<matplotlib.lines.Line2D at 0x7f2fe0051370>,
 <matplotlib.lines.Line2D at 0x7f2fe0051460>,
 <matplotlib.lines.Line2D at 0x7f2fe0051310>,
 <matplotlib.lines.Line2D at 0x7f2fe0051670>]
```




1
0
−1
−2
−3
40
50
60
70
80
90
100
110
120

Practical 5

Write a program to implement linear Support Vector Machine algorithm using a set of training data samples.

In [17]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [18]:

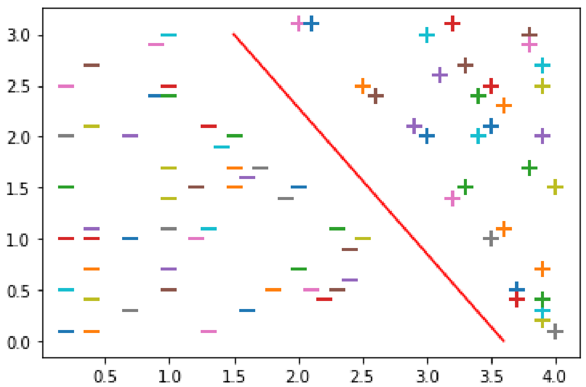
```
X = np.array([[1.6,0.3], [1.8,0.5], [2.0,0.7], [2.2,0.4], [2.4,0.6], [2.3,0.5], [2.1,0.5],
              [1.7,1.7], [2.5,1.0], [1.0,3.0], [2.0,1.5], [1.5,1.5], [1.5,2.0], [1.0,2.5],
              [1.6,1.6], [2.4,0.9], [0.9,2.9], [1.9,1.4], [1.0,1.4], [1.4,1.9], [0.9,2.4],
              [1.5,1.7], [2.3,1.1], [0.4,1.0], [1.0,0.7], [1.2,1.5], [1.2,1.0], [1.0,1.1],
              [1.0,1.7], [1.3,1.1], [0.7,1.0], [0.4,0.7], [0.2,1.5], [0.2,1.0], [0.4,1.1],
              [1.0,0.5], [1.3,0.1], [0.7,0.3], [0.4,0.4], [0.2,0.5], [0.2,0.1], [0.4,0.1],
              [1.0,2.4], [1.3,2.1], [0.7,2.0], [0.4,2.7], [0.2,2.5], [0.2,2.0], [0.4,2.1],
              [3.4,2.0], [3.5,2.1], [3.6,2.3], [3.4,2.4], [3.5,2.5], [3.1,2.6], [3.3,2.7],
              [2.0,3.1], [3.5,1.0], [4.0,1.5], [3.0,3.0], [3.0,2.0], [2.5,2.5], [3.3,1.5],
              [3.9,2.5], [3.9,2.0], [3.8,3.0], [3.8,2.9], [3.9,2.7], [3.9,2.5], [3.9,2.7],
              [2.1,3.1], [3.6,1.1], [3.8,1.7], [3.2,3.1], [2.9,2.1], [2.6,2.4], [3.2,1.4],
              [4.0,0.1], [3.9,0.2], [3.9,0.3], [3.7,0.5], [3.9,0.7], [3.9,0.4], [3.7,0.4]])


Y = np.array([-1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              1, 1, 1, 1, 1, 1, 1,
              1, 1, 1, 1, 1, 1, 1,
              1, 1, 1, 1, 1, 1, 1,
              1, 1, 1, 1, 1, 1, 1,
              1, 1, 1, 1, 1, 1, 1])
```

In [19]:

```
for i in range(len(X)):
    if Y[i] == -1:
        plt.scatter(X[i][0], X[i][1], s=120, marker='_', linewidths=2)
    else:
        plt.scatter(X[i][0], X[i][1], s=120, marker='+', linewidths=2)

plt.plot([3.6,1.5],[0.0,3.0],'r')
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x7fe630874460>]
```



In [20]:

```
def train_svm(X, Y, epochs=10000):
    w = np.zeros(len(X[0]))

    learning_rate = 1

    w0_per_epoch = []
    w1_per_epoch = []

    print("starts training")
    for epoch in range(1, epochs):
        error = 0
        for i, x in enumerate(X):
            if (Y[i] * np.dot(X[i], w)) < 1:
                w = w + learning_rate * ((X[i] * Y[i]) + (-2 * (1/epochs) * w))
            else:
                w = w + learning_rate * (-2 * (1/epochs) * w)

        w0_per_epoch.append(w[0])
        w1_per_epoch.append(w[1])

    return w, w0_per_epoch, w1_per_epoch
```

In [21]:

```
w, w0array, w1array = train_svm(X, Y, epochs=10000)
print(w)
```

```
starts training
[3.36748683 2.10292688]
```

In [22]:

```
epochs = len(w0array)

number_of_weights_to_graph = 100

num_per_epoch = epochs/number_of_weights_to_graph

w0_to_graph = []
w1_to_graph = []
epoch_to_graph = []

for i in range(number_of_weights_to_graph):
    epoch_to_graph.append(int(num_per_epoch*i))
    w0_to_graph.append(w0array[int(num_per_epoch*i)])
    w1_to_graph.append(w1array[int(num_per_epoch*i)])
```

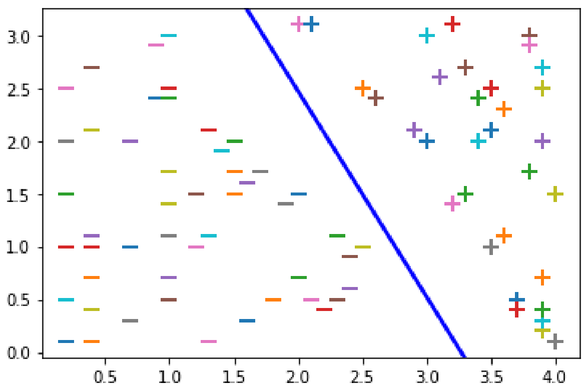
In [23]:

```
X = np.array([[1.6,0.3], [1.8,0.5], [2.0,0.7], [2.2,0.4], [2.4,0.6], [2.3,0.5], [2.1,0.5],
              [1.7,1.7], [2.5,1.0], [1.0,3.0], [2.0,1.5], [1.5,1.5], [1.5,2.0], [1.0,2.5],
              [1.6,1.6], [2.4,0.9], [0.9,2.9], [1.9,1.4], [1.0,1.4], [1.4,1.9], [0.9,2.4],
              [1.5,1.7], [2.3,1.1], [0.4,1.0], [1.0,0.7], [1.2,1.5], [1.2,1.0], [1.0,1.1],
              [1.0,1.7], [1.3,1.1], [0.7,1.0], [0.4,0.7], [0.2,1.5], [0.2,1.0], [0.4,1.1],
              [1.0,0.5], [1.3,0.1], [0.7,0.3], [0.4,0.4], [0.2,0.5], [0.2,0.1], [0.4,0.1],
              [1.0,2.4], [1.3,2.1], [0.7,2.0], [0.4,2.7], [0.2,2.5], [0.2,2.0], [0.4,2.1],
              [3.4,2.0], [3.5,2.1], [3.6,2.3], [3.4,2.4], [3.5,2.5], [3.1,2.6], [3.3,2.7],
              [2.0,3.1], [3.5,1.0], [4.0,1.5], [3.0,3.0], [3.0,2.0], [2.5,2.5], [3.3,1.5],
              [3.9,2.5], [3.9,2.0], [3.8,3.0], [3.8,2.9], [3.9,2.7], [3.9,2.5], [3.9,2.7],
              [2.1,3.1], [3.6,1.1], [3.8,1.7], [3.2,3.1], [2.9,2.1], [2.6,2.4], [3.2,1.4],
              [4.0,0.1], [3.9,0.2], [3.9,0.3], [3.7,0.5], [3.9,0.7], [3.9,0.4], [3.7,0.4]])

Y = np.array([-1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
              -1, -1, -1, -1, -1, -1, -1,
```

```
               1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1])


for i in range(len(X)):
    if Y[i] == -1:
        plt.scatter(X[i][0], X[i][1], s=120, marker='_', linewidths=2)
    else:
        plt.scatter(X[i][0], X[i][1], s=120, marker='+', linewidths=2)

x2=[w[0]*0.65,w[1],-w[1],w[0]]
x3=[w[0]*0.65,w[1],w[1],-w[0]]

x2x3 =np.array([x2,x3])
X,Y,U,V = zip(*x2x3)
ax = plt.gca()
ax.quiver(X,Y,U,V,scale=1, color='blue')
```

Out[23]:

```
<matplotlib.quiver.Quiver at 0x7fe6307e3e80>
```



Practical 6

Write A Program In Python To Implement ID3.

In [3]:

```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log

df = pd.read_csv('dataset.csv')
df
```

Out[3]:

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rainy	Mild	High	Weak	Yes
4	Rainy	Cold	Normal	Weak	Yes
5	Rainy	Cold	Normal	Strong	No
6	Overcast	Cold	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cold	Normal	Weak	Yes
9	Rainy	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rainy	Mild	High	Strong	No

In [4]:

```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy
```

In [5]:

```
def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]
  target_variables = df[Class].unique()
  variables = df[attribute].unique()
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
          den = len(df[attribute][df[attribute]==variable])
          fraction = num/(den+eps)
          entropy += -fraction*log(fraction+eps)
      fraction2 = den/len(df)
      entropy2 += -fraction2*entropy
  return abs(entropy2)
```

In [6]:

```
def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]
```

In [7]:

```
def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)
```

In [8]:

```
def buildTree(df,tree=None):
    Class = df.keys()[-1]
    node = find_winner(df)

    attValue = np.unique(df[node])
    if tree is None:
        tree={}
        tree[node] = {}
    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['PlayTennis'],return_counts=True)


        if len(counts)==1:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)
    return tree
```

In [15]:

```
t = buildTree(df)

for i,c in t.items():
    print(i)
    for j in c.items():
        print(j)
```

```
Outlook
('Overcast', 'Yes')
('Rainy', {'Windy': {'Strong': 'No', 'Weak': 'Yes'}})
('Sunny', {'Humidity': {'High': 'No', 'Normal': 'Yes'}})
```

Practical 7 (a)

Implement A Perceptron For Binary AND Operation

In [33]:

```
import numpy as np
```

In [34]:

```
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
```

In [35]:

```
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
```

In [36]:

```
def AND_Logic(x):
    w = np.array([0.5, 0.5])
    b = -1
    return perceptronModel(x, w, b)
```

In [37]:

```
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
```

In [38]:

```
print("AND ({}, {}) = {}".format(0, 0, AND_Logic(test1)))
print("AND ({}, {}) = {}".format(0, 1, AND_Logic(test2)))
print("AND ({}, {}) = {}".format(1, 0, AND_Logic(test3)))
print("AND ({}, {}) = {}".format(1, 1, AND_Logic(test4)))
```

```
AND (0, 0) = 0
AND (0, 1) = 0
AND (1, 0) = 0
AND (1, 1) = 1
```

In [ ]:

```

```

Practical 7 (b)

Implement A Perceptron For Binary OR Operation

In [39]:

```
import numpy as np
```

In [40]:

```
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
```

In [41]:

```
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
```

In [42]:

```
def OR_Logic(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptronModel(x, w, b)
```

In [43]:

```
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])
```

In [44]:

```
print("OR ({}, {}) = {}".format(0, 0, OR_Logic(test1)))
print("OR ({}, {}) = {}".format(0, 1, OR_Logic(test2)))
print("OR ({}, {}) = {}".format(1, 0, OR_Logic(test3)))
print("OR ({}, {}) = {}".format(1, 1, OR_Logic(test4)))
```

```
OR (0, 0) = 0
OR (0, 1) = 1
OR (1, 0) = 1
OR (1, 1) = 1
```

Practical 8

Write A Program In Python To Implement Back Propagation Algorithm.

In [3]:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # two inputs [sleep,study]
y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in Exams]
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100
```

In [5]:

```
def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)
```

In [7]:

```
epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

In [9]:

```
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

In [13]:

```
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)


    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad


    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
```

In [15]:

```
print("Input: " + str(X))
print("\nActual Output: " + str(y))
print("\nPredicted Output: " ,output)
```

```
Input: [[0.66666667 1.        ]
```

```
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output: [[0.92]
 [0.86]
 [0.89]]

Predicted Output:  [[0.89590721]
 [0.87585879]
 [0.89760529]]
```

Practical 9

Write A Program In Python To Implement Naive Bayes Theorem

In [23]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
```

In [24]:

```
def accuracy_score(y_true, y_pred):

    return round(float(sum(y_pred == y_true))/float(len(y_true)) * 100 ,2)

def pre_processing(df):

    X = df.drop([df.columns[-1]], axis = 1)
    y = df[df.columns[-1]]

    return X, y
```

In [25]:

```
class  NaiveBayes:

    def __init__(self):
        self.features = list
        self.likelihoods = {}
        self.class_priors = {}
        self.pred_priors = {}

        self.X_train = np.array
        self.y_train = np.array
        self.train_size = int
        self.num_feats = int

    def fit(self, X, y):
        self.features = list(X.columns)
        self.X_train = X
        self.y_train = y
        self.train_size = X.shape[0]
        self.num_feats = X.shape[1]

        for feature in self.features:
            self.likelihoods[feature] = {}
            self.pred_priors[feature] = {}

            for feat_val in np.unique(self.X_train[feature]):
                self.pred_priors[feature].update({feat_val: 0})

                for outcome in np.unique(self.y_train):
                    self.likelihoods[feature].update({feat_val+'_'+outcome:0})
                    self.class_priors.update({outcome: 0})

        self._calc_class_prior()
        self._calc_likelihoods()
        self._calc_predictor_prior()

    def _calc_class_prior(self):
        for outcome in np.unique(self.y_train):
            outcome_count = sum(self.y_train == outcome)
            self.class_priors[outcome] = outcome_count / self.train_size

    def _calc_likelihoods(self):
        for feature in self.features:

            for outcome in np.unique(self.y_train):
```

```
            for outcome in np.unique(self.y_train):
                outcome_count = sum(self.y_train == outcome)
                feat_likelihood = self.X_train[feature][self.y_train[self.y_train == outcome].index
.values.tolist()].value_counts().to_dict()

                for feat_val, count in feat_likelihood.items():
                    self.likelihoods[feature][feat_val + '_' + outcome] = count/outcome_count

    def _calc_predictor_prior(self):
        for feature in self.features:
            feat_vals = self.X_train[feature].value_counts().to_dict()

            for feat_val, count in feat_vals.items():
                self.pred_priors[feature][feat_val] = count/self.train_size


    def predict(self, X):
        results = []
        X = np.array(X)

        for query in X:
            probs_outcome = {}
            for outcome in np.unique(self.y_train):
                prior = self.class_priors[outcome]
                likelihood = 1
                evidence = 1

                for feat, feat_val in zip(self.features, query):
                    likelihood *= self.likelihoods[feat][feat_val + '_' + outcome]
                    evidence *= self.pred_priors[feat][feat_val]

                posterior = (likelihood * prior) / (evidence)
                probs_outcome[outcome] = posterior

            result = max(probs_outcome, key = lambda x: probs_outcome[x])
            results.append(result)

        return np.array(results)
```

In [27]:

```
df = pd.read_csv("dataset.csv")

X,y  = pre_processing(df)

nb_clf = NaiveBayes()
nb_clf.fit(X, y)

print("Train Accuracy: {}".format(accuracy_score(y, nb_clf.predict(X))))

query = np.array([['Rainy','Mild', 'Normal', 'T']])
print("Query 1:- {} ---> {}".format(query, nb_clf.predict(query)))

query = np.array([['Overcast','Cold', 'Normal', 'T']])
print("Query 2:- {} ---> {}".format(query, nb_clf.predict(query)))

query = np.array([['Sunny','Hot', 'High', 'T']])
print("Query 3:- {} ---> {}".format(query, nb_clf.predict(query)))
```

```
Train Accuracy: 92.86
Query 1:- [['Rainy' 'Mild' 'Normal' 'T']] ---> ['Yes']
Query 2:- [['Overcast' 'Cold' 'Normal' 'T']] ---> ['Yes']
Query 3:- [['Sunny' 'Hot' 'High' 'T']] ---> ['No']
```

Practical 10

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

In [2]:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

In [4]:

```
df = pd.read_csv('dataset.csv')
df = df.replace('?',np.nan)
df
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	45	1	1	110	264	0	0	132	0	1.2	2	0	7	1
299	68	1	4	144	193	1	0	141	0	3.4	2	2	7	2
300	57	1	4	130	131	0	0	115	1	1.2	2	1	7	3
301	57	0	2	130	236	0	2	174	0	0.0	2	1	3	1
302	38	1	3	138	175	0	0	173	0	0.0	1	NaN	3	0

303 rows × 14 columns

In [5]:

```
print('\n Attributes and datatypes')
print(df.dtypes)
```

```
 Attributes and datatypes
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object
```

In [ ]:

```
model =  BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp
','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
```

In [ ]:

```
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(df,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Practical 11

Develop A Genetic Algorithm For Optimization Of Hyper Parameters In Machine Learning.

In [1]:

```
import random
import numpy as np
DATA = [[3, 7],# Benefit, Weight
        [8, 8],
        [3, 4],
        [2, 10],
        [7, 4],
        [9, 6],
        [4, 4]]
INDIVIDUAL_NAMES = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```

In [2]:

```
class Individual:
    def __init__(self, name, encoding):
        self.name = name
        self.encoding = encoding
        self.weight = self.getWeight()
        self.fitness = self.getFitness()
        self.probability = 0
        self.cumProb = 0

    def getWeight(self):
        weightTemp = 0
        for i in range(len(self.encoding)):
            if self.encoding[i] == 1:
                weightTemp += DATA[i][1]
        return weightTemp

    def getFitness(self):
        fitnessTemp = 0
        for i in range(len(self.encoding)):
            if self.encoding[i] == 1:
                fitnessTemp += DATA[i][0]
        if self.weight <= 22:
            return fitnessTemp
        else:
            return 0

    def getEncoding(self):
        return self.encoding

    def setProbability(self, fitness, fitnessCumulative):
        self.probability = fitness/fitnessCumulative

    def setCumProb(self, cumProb):
        self.cumProb = cumProb
```

In [3]:

```
MAX_LOOP = 10
initialPop = True
totalFitnesses = []
```

In [4]:

```
for gen in range(MAX_LOOP):

    print('################### GENERATION NO', gen, '###################')
    print()
```

```
    print('### New individuals gen', gen, '###')
    INDIVIDUALS = [None for _ in range(len(INDIVIDUAL_NAMES))]
    if initialPop == True:
        random.seed(13)
        for i in range(len(INDIVIDUALS)):
            INDIVIDUALS[i] = Individual(INDIVIDUAL_NAMES[i], [random.randrange(2) for _ in range(len
(DATA))])
            print(INDIVIDUALS[i].name, INDIVIDUALS[i].encoding,
                INDIVIDUALS[i].weight, INDIVIDUALS[i].fitness)
        print()
        initialPop = False

    else:
        INDIVIDUALS = [None for _ in range(len(INDIVIDUAL_NAMES))]
        for i in range(len(INDIVIDUALS)):
            INDIVIDUALS[i] = Individual(INDIVIDUAL_NAMES[i], ENCODINGS[i])
            print(INDIVIDUALS[i].name, INDIVIDUALS[i].encoding,
                INDIVIDUALS[i].weight, INDIVIDUALS[i].fitness)
        print()

    print('### Sorted individuals gen', gen, '###')
    INDIVIDUALS.sort(key=lambda x: x.fitness, reverse=True)

    for i in range(len(INDIVIDUALS)):
        print(INDIVIDUALS[i].name, INDIVIDUALS[i].encoding,
            INDIVIDUALS[i].weight, INDIVIDUALS[i].fitness)

    print()

    print('### Probability of individuals gen', gen, '###')
    fitnessCumulative = 0
    for i in range(len(INDIVIDUALS)):
        fitnessCumulative += INDIVIDUALS[i].fitness

    for i in range(len(INDIVIDUALS)):
        INDIVIDUALS[i].setProbability(INDIVIDUALS[i].fitness, fitnessCumulative)
    for i in range(len(INDIVIDUALS)):
        print(INDIVIDUALS[i].name, INDIVIDUALS[i].encoding,
            INDIVIDUALS[i].weight, INDIVIDUALS[i].fitness, INDIVIDUALS[i].probability)
    totalFitnesses.append(fitnessCumulative)
    print()

    print('### Cumulative probability of individuals gen', gen, '###')
    cumProb = 0
    for i in range(len(INDIVIDUALS)):
        cumProb += INDIVIDUALS[i].probability
        INDIVIDUALS[i].setCumProb(cumProb)
    for i in range(len(INDIVIDUALS)):
        print(INDIVIDUALS[i].name, INDIVIDUALS[i].cumProb)
    print()

    random.seed(13)
    randomRWS = []
    for i in range(len(INDIVIDUALS)):
        randomRWS.append(random.random())

    print('### Generated random values to perform RWS selection gen', gen, '###')
    for i in range(len(randomRWS)):
        print(randomRWS[i])

    print()

    resultRWS = []
    for i in range(len(randomRWS)):
        for j in range(len(INDIVIDUALS)):
            if randomRWS[i] < INDIVIDUALS[j].cumProb:
                resultRWS.append(INDIVIDUALS[j])
                break
            else:
                continue

    print('### Selected individuals gen', gen, '###')
    for i in range(len(resultRWS)):
        print(resultRWS[i].name, resultRWS[i].fitness, resultRWS[i].encoding)

    print()
```

```
    resultRWS.sort(key=lambda x: x.fitness, reverse=True)

    print('### Sorted selected individuals based on fitness gen', gen, '(the two best individuals will not be crossovered)###')
    for i in range(len(resultRWS)):
        print(resultRWS[i].name, resultRWS[i].fitness, resultRWS[i].encoding)

    print()

    resultRWScopy = resultRWS[2:]
    random.shuffle(resultRWScopy)
    resultRWS[2:] = resultRWScopy

    print('### Multipoint crossover (index 0, 2, 4, 6) gen', gen, '###')
    ENCODINGS = [resultRWS[i].encoding for i in range(2)]

    CROSSOVER = [resultRWS[i].encoding for i in range(2,len(resultRWS[0].encoding)+1)]
    CROSSOVER = np.array(CROSSOVER)
    CROSSOVER = CROSSOVER.tolist()


    print("Before crossover")
    for i in range(len(CROSSOVER)):
        print(CROSSOVER[i])
    print()

    for i in range(0,len(CROSSOVER),2):
        for j in range(0,len(CROSSOVER[0]),2):
            temp = CROSSOVER[i][j]
            CROSSOVER[i][j] = CROSSOVER[i+1][j]
            CROSSOVER[i+1][j] = temp

    print("After crossover")
    for i in range(len(CROSSOVER)):
        print(CROSSOVER[i])
    print()

    ENCODINGS += CROSSOVER
    print('### Elitism individuals + crossovered individuals gen', gen, '###')
    for i in range(len(ENCODINGS)):
        print(ENCODINGS[i])
    print()
```

```
################### GENERATION NO 0 ####################

### New individuals gen 0 ###
A [1, 1, 0, 0, 0, 0, 0] 15 11
B [0, 0, 0, 1, 0, 1, 0] 16 11
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 1, 0, 1, 1, 1] 26 0
E [1, 1, 0, 1, 1, 1, 0] 35 0
F [1, 1, 0, 1, 0, 0, 1] 29 0
G [1, 1, 1, 1, 1, 1, 0] 39 0
H [1, 0, 0, 0, 1, 0, 0] 11 10

### Sorted individuals gen 0 ###
C [0, 1, 0, 0, 1, 1, 1] 22 28
A [1, 1, 0, 0, 0, 0, 0] 15 11
B [0, 0, 0, 1, 0, 1, 0] 16 11
H [1, 0, 0, 0, 1, 0, 0] 11 10
D [0, 1, 1, 0, 1, 1, 1] 26 0
E [1, 1, 0, 1, 1, 1, 0] 35 0
F [1, 1, 0, 1, 0, 0, 1] 29 0
G [1, 1, 1, 1, 1, 1, 0] 39 0

### Probability of individuals gen 0 ###
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.4666666666666667
A [1, 1, 0, 0, 0, 0, 0] 15 11 0.18333333333333332
B [0, 0, 0, 1, 0, 1, 0] 16 11 0.18333333333333332
H [1, 0, 0, 0, 1, 0, 0] 11 10 0.16666666666666666
D [0, 1, 1, 0, 1, 1, 1] 26 0 0.0
E [1, 1, 0, 1, 1, 1, 0] 35 0 0.0
F [1, 1, 0, 1, 0, 0, 1] 29 0 0.0
G [1, 1, 1, 1, 1, 1, 0] 39 0 0.0

### Cumulative probability of individuals gen 0 ###
```

```
C 0.4666666666666667
A 0.65
B 0.8333333333333334
H 1.0
D 1.0
E 1.0
F 1.0
G 1.0

### Generated random values to perform RWS selection gen 0 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 0 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
B 11 [0, 0, 0, 1, 0, 1, 0]
B 11 [0, 0, 0, 1, 0, 1, 0]
H 10 [1, 0, 0, 0, 1, 0, 0]
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 0 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]
C 28 [0, 1, 0, 0, 1, 1, 1]
B 11 [0, 0, 0, 1, 0, 1, 0]
B 11 [0, 0, 0, 1, 0, 1, 0]
H 10 [1, 0, 0, 0, 1, 0, 0]

### Multipoint crossover (index 0, 2, 4, 6) gen 0 ###
Before crossover
[0, 0, 0, 1, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 0, 0, 1, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]
[1, 0, 0, 0, 1, 0, 0]

After crossover
[0, 0, 0, 1, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 0, 1, 1, 0]
[0, 0, 0, 0, 1, 0, 1]

### Elitism individuals + crossovered individuals gen 0 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 0, 0, 1, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 0, 1, 1, 0]
[0, 0, 0, 0, 1, 0, 1]

#################### GENERATION NO 1 ####################

### New individuals gen 1 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 0, 0, 1, 1, 1, 1] 24 0
D [0, 1, 0, 0, 0, 1, 0] 14 17
E [0, 1, 0, 0, 0, 1, 0] 14 17
F [0, 0, 0, 1, 1, 1, 1] 24 0
G [1, 1, 0, 0, 1, 1, 0] 25 0
```

```
H [0, 0, 0, 0, 1, 0, 1] 8 11

### Sorted individuals gen 1 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 0, 1, 0] 14 17
E [0, 1, 0, 0, 0, 1, 0] 14 17
H [0, 0, 0, 0, 1, 0, 1] 8 11
C [0, 0, 0, 1, 1, 1, 1] 24 0
F [0, 0, 0, 1, 1, 1, 1] 24 0
G [1, 1, 0, 0, 1, 1, 0] 25 0

### Probability of individuals gen 1 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.27722772277227725
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.27722772277227725
D [0, 1, 0, 0, 0, 1, 0] 14 17 0.16831683168316833
E [0, 1, 0, 0, 0, 1, 0] 14 17 0.16831683168316833
H [0, 0, 0, 0, 1, 0, 1] 8 11 0.10891089108910891
C [0, 0, 0, 1, 1, 1, 1] 24 0 0.0
F [0, 0, 0, 1, 1, 1, 1] 24 0 0.0
G [1, 1, 0, 0, 1, 1, 0] 25 0 0.0

### Cumulative probability of individuals gen 1 ###
A 0.27722772277227725
B 0.5544554455445545
D 0.7227722772277229
E 0.8910891089108912
H 1.0000000000000002
C 1.0000000000000002
F 1.0000000000000002
G 1.0000000000000002

### Generated random values to perform RWS selection gen 1 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 1 ###
A 28 [0, 1, 0, 0, 1, 1, 1]
D 17 [0, 1, 0, 0, 0, 1, 0]
D 17 [0, 1, 0, 0, 0, 1, 0]
E 17 [0, 1, 0, 0, 0, 1, 0]
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 1 (the two best individuals will not be cross
overed)###
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]
A 28 [0, 1, 0, 0, 1, 1, 1]
D 17 [0, 1, 0, 0, 0, 1, 0]
D 17 [0, 1, 0, 0, 0, 1, 0]
E 17 [0, 1, 0, 0, 0, 1, 0]

### Multipoint crossover (index 0, 2, 4, 6) gen 1 ###
Before crossover
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]
```

```
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 1 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 2 ####################

### New individuals gen 2 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 0, 1, 0] 14 17
E [0, 1, 0, 0, 0, 1, 0] 14 17
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 0, 1, 0] 14 17
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 2 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 0, 1, 0] 14 17
E [0, 1, 0, 0, 0, 1, 0] 14 17
G [0, 1, 0, 0, 0, 1, 0] 14 17

### Probability of individuals gen 2 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.14659685863874344
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.14659685863874344
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.14659685863874344
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.14659685863874344
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.14659685863874344
D [0, 1, 0, 0, 0, 1, 0] 14 17 0.08900523560209424
E [0, 1, 0, 0, 0, 1, 0] 14 17 0.08900523560209424
G [0, 1, 0, 0, 0, 1, 0] 14 17 0.08900523560209424

### Cumulative probability of individuals gen 2 ###
A 0.14659685863874344
B 0.2931937172774869
C 0.43979057591623033
F 0.5863874345549738
H 0.7329842931937172
D 0.8219895287958114
E 0.9109947643979057
G 1.0

### Generated random values to perform RWS selection gen 2 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 2 ###
B 28 [0, 1, 0, 0, 1, 1, 1]
H 28 [0, 1, 0, 0, 1, 1, 1]
H 28 [0, 1, 0, 0, 1, 1, 1]
E 17 [0, 1, 0, 0, 0, 1, 0]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 2 (the two best individuals will not be cross
overed)###
```

```
B 28 [0, 1, 0, 0, 1, 1, 1]
H 28 [0, 1, 0, 0, 1, 1, 1]
H 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
E 17 [0, 1, 0, 0, 0, 1, 0]

### Multipoint crossover (index 0, 2, 4, 6) gen 2 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 2 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 3 ####################

### New individuals gen 3 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 0, 1, 0] 14 17
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 3 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 0, 1, 0] 14 17

### Probability of individuals gen 3 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.13145539906103287
G [0, 1, 0, 0, 0, 1, 0] 14 17 0.07981220657276995

### Cumulative probability of individuals gen 3 ###
A 0.13145539906103287
B 0.26291079812206575
C 0.3943661971830986
D 0.5258215962441315
E 0.6572769953051644
F 0.7887323943661972
H 0.9201877934272301
G 1.0
```

```
### Generated random values to perform RWS selection gen 3 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 3 ###
B 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
H 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 3 (the two best individuals will not be cross
overed)###
B 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
H 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Multipoint crossover (index 0, 2, 4, 6) gen 3 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 3 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

################### GENERATION NO 4 ####################

### New individuals gen 4 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 4 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
```

```
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Probability of individuals gen 4 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
G [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.125

### Cumulative probability of individuals gen 4 ###
A 0.125
B 0.25
C 0.375
D 0.5
E 0.625
F 0.75
G 0.875
H 1.0

### Generated random values to perform RWS selection gen 4 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 4 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 4 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Multipoint crossover (index 0, 2, 4, 6) gen 4 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 4 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
```

```
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 5 ####################

### New individuals gen 5 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 5 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Probability of individuals gen 5 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
G [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.125

### Cumulative probability of individuals gen 5 ###
A 0.125
B 0.25
C 0.375
D 0.5
E 0.625
F 0.75
G 0.875
H 1.0

### Generated random values to perform RWS selection gen 5 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 5 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 5 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
```

```
### Multipoint crossover (index 0, 2, 4, 6) gen 5 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 5 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 6 ####################

### New individuals gen 6 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 6 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Probability of individuals gen 6 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
G [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.125

### Cumulative probability of individuals gen 6 ###
A 0.125
B 0.25
C 0.375
D 0.5
E 0.625
F 0.75
G 0.875
H 1.0

### Generated random values to perform RWS selection gen 6 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
```

```
0.14715991816841778
0.22516293556211264

### Selected individuals gen 6 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 6 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Multipoint crossover (index 0, 2, 4, 6) gen 6 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 6 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 7 ####################

### New individuals gen 7 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 7 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Probability of individuals gen 7 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
```

```
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
G [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.125

### Cumulative probability of individuals gen 7 ###
A 0.125
B 0.25
C 0.375
D 0.5
E 0.625
F 0.75
G 0.875
H 1.0

### Generated random values to perform RWS selection gen 7 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 7 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 7 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Multipoint crossover (index 0, 2, 4, 6) gen 7 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 7 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 8 ####################

### New individuals gen 8 ###
```

```
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 8 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Probability of individuals gen 8 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
G [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.125

### Cumulative probability of individuals gen 8 ###
A 0.125
B 0.25
C 0.375
D 0.5
E 0.625
F 0.75
G 0.875
H 1.0

### Generated random values to perform RWS selection gen 8 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 8 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 8 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Multipoint crossover (index 0, 2, 4, 6) gen 8 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
```

```
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 8 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

#################### GENERATION NO 9 ####################

### New individuals gen 9 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Sorted individuals gen 9 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28
B [0, 1, 0, 0, 1, 1, 1] 22 28
C [0, 1, 0, 0, 1, 1, 1] 22 28
D [0, 1, 0, 0, 1, 1, 1] 22 28
E [0, 1, 0, 0, 1, 1, 1] 22 28
F [0, 1, 0, 0, 1, 1, 1] 22 28
G [0, 1, 0, 0, 1, 1, 1] 22 28
H [0, 1, 0, 0, 1, 1, 1] 22 28

### Probability of individuals gen 9 ###
A [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
B [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
C [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
D [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
E [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
F [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
G [0, 1, 0, 0, 1, 1, 1] 22 28 0.125
H [0, 1, 0, 0, 1, 1, 1] 22 28 0.125

### Cumulative probability of individuals gen 9 ###
A 0.125
B 0.25
C 0.375
D 0.5
E 0.625
F 0.75
G 0.875
H 1.0

### Generated random values to perform RWS selection gen 9 ###
0.2590084917154736
0.6852579929645369
0.6840819180161107
0.8493361613899302
0.1857241738737354
0.2305586089654681
0.14715991816841778
0.22516293556211264

### Selected individuals gen 9 ###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
```

```
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Sorted selected individuals based on fitness gen 9 (the two best individuals will not be cross
overed)###
C 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
F 28 [0, 1, 0, 0, 1, 1, 1]
G 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]
B 28 [0, 1, 0, 0, 1, 1, 1]

### Multipoint crossover (index 0, 2, 4, 6) gen 9 ###
Before crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

After crossover
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]

### Elitism individuals + crossovered individuals gen 9 ###
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 1]
```